Modelos de programación 2

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Modelos de programación 2 / Gustavo Enrique Tabares Parra, /
Bogotá D.C., Fundación Universitaria del Área Andina. 2017

978-958-5459-35-9

Catalogación en la fuente Fundación Universitaria del Área Andina (Bogotá).

Edición:
Fondo editorial Areandino
Fundación Universitaria del Área Andina
Calle 71 11-14, Bogotá D.C., Colombia
Tel.: (57-1) 7 42 19 64 ext. 1228
E-mail: publicaciones@areandina.edu.co
http://www.areandina.edu.co

Primera edición: noviembre de 2017

Corrección de estilo, diagramación y edición: Dirección Nacional de Operaciones virtuales
Diseño y compilación electrónica: Dirección Nacional de Investigación

Hecho en Colombia
Made in Colombia

Modelos de programación 2

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Índice

UNIDAD 1 Java web Servlet y JSP

Introducción 7

Metodología 8

Desarrollo temático 9

UNIDAD 1 Arquitectura Modelo- Vista-Controlador

Introducción 23

Metodología 24

Desarrollo temático 25

UNIDAD 2 Java Sever Faces Tecnología

Introducción 42

Metodología 43

Desarrollo temático 44

UNIDAD 2 Servicio web

Introducción 58

Metodología 59

Desarrollo temático 60

Índice

UNIDAD 3 Beans Enterprise

Introducción 81

Metodología 82

Desarrollo temático 83

UNIDAD 3 Java persistence API JPA

Introducción 101

Metodología 102

Desarrollo temático 103

UNIDAD 4 Seguridad web

Introducción 128

Metodología 129

Desarrollo temático 130

UNIDAD 4 ISeguridad en aplicaciones web

Introducción 139

Metodología 140

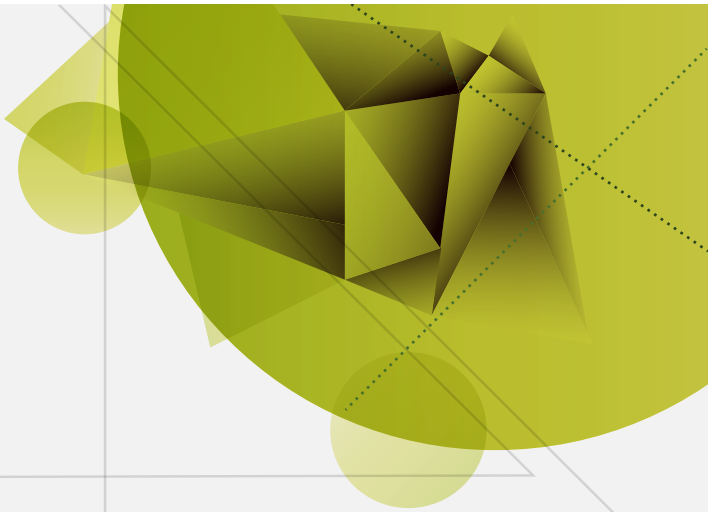
Desarrollo temático 141

Bibliografía 153

1

Unidad 1

Java web Servlet y JSP



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Aquí se debe explicar los alcances y propósitos generales de la cartilla y hacer referencia a los temas que lo integran, se debe mostrar por qué los temas serán útiles e interesantes, puede hacerse desde la aplicación del tema a situaciones reales.

U1

Metodología

Aquí se deben presentar orientaciones metodológicas: cómo leer, comprender y asimilar de mejor manera los contenidos.

U1 Desarrollo temático

Java web Servlet y JSP

En esta cartilla el estudiante aprenderá a usar Java para el desarrollo de aplicaciones web, aplicaciones que están del lado del servidor y nos basaremos en las tecnologías como Servlet y JSP Java Server Pages.

El desarrollo de aplicaciones web con Servlet es usado en la mayor parte del mercado de las empresas, donde se aplica esta tecnología de desarrollo, como son: El gobierno, entidades financieras y grandes empresas tienen bajo se organización el servicio de aplicaciones bajo esta arquitectura de software en el API de Servlet.

Para el desarrollo de esta unidad se requiere contar con algunas herramientas de protocolos http, manejo de HTML, comprender el paradigma cliente servidor, el protocolo TCP/IP y programar en el lenguaje Java.

Los Servlet

Es un programa que se ejecuta en un servidor web, construyendo una página web para ser devuelta al usuario, se construye dinámicamente conteniendo información procedente de bases de datos, respuestas a peticiones mediante datos introducidos por el usuario, los Servlet en Java es una clase que presenta una serie de ventajas a las anteriores CGI que se desarrollaban inicialmente, presentando ciertas ventajas como la portabilidad, son más potentes, mucho más eficientes y más fáciles de usar.

Un contenedor "controlador" Servlet es un programa en Java que recibe las peticiones de una página web y redireccionan estas peticiones a un objeto Servlet.

El modelo de Servlet la máquina virtual de Java siendo el entorno de ejecución de los Servlet se arrancar al iniciar el servidor, permaneciendo activo durante todo la ejecución de la aplicación con el objetivo de atender cada petición o proceso solicitado por el usuario, creando un hilo (thread) procesos muchos más rápidos.

Las páginas están basadas en informar e interactuar con los usuarios, como ejemplo encontramos los motores de búsqueda dando respuesta a los diferentes peticiones que generan los usuarios, Las aplicaciones web que procesan pedidos desde sus sitios web a través del comercio electrónico. Los Servlet ofrecen una infraestructura en el desarrollo de aplicaciones web, proporcionando métodos para el análisis automático y de codificación de los datos de formularios html.

En constante cambio que enfrenta la información, por ejemplo el cambio de dólar, la información que se da con el cambio del cli-

ma, o noticias que cambian dinámicamente en la cabecera de las páginas web, en muchas ocasiones construyendo la página con el resultado de la consulta que el cliente solicito.

También encontramos información que usan las páginas web desde Bases de Datos corporativas u otras fuentes, el uso de la información a través de una tienda en línea, la información de los productos debe estas almacenada en un BD, para que los clientes puedan interactuar con ella. También la información del cliente es importante almacenarla para llevar un control de quienes adquieren productos en línea.

El acceso a las cabeceras de las peticiones http a través de dos tipos get y post, donde el post viene por el cuerpo de la petición, conteniendo ficheros y el GET la petición viene por la cabecera URL.

Los Servlet en Java permiten realizar muchas más cosas que en la programación tradicional es imposible, los Servlet permiten compartir datos entre sí, conexiones a bases de datos, ficheros, etc.

Ventajas de los Servlets

Eficiencia: cada petición por parte de un cliente crea un hilo, no un nuevo proceso como ocurría con los CGI tradicionales, donde se presentaba la sobrecarga de procesos de arracada y dominaba el tiempo de ejecución. Los Servlets, la máquina virtual de Java permanece activada corriendo, y cada petición de la aplicación se maneja como un thread Java muy liviano y no como un proceso del sistema que requiere memoria de ejecución para su procesamiento. Los Servlet también tienen más alternativas que los programas normales CGI para optimizaciones como los cachés de cálculos previos, mantener abiertas las conexiones de bases de datos, etc.

Potencia: con los Servlets se controla fácilmente las deficiencias que su presentaron anteriormente en el desarrollo de los CGI al ejecutarse, Los Servlets mantiene las una comunicación directa entre la aplicación y el servidor web. Pueden compartir datos entre ellos. Como puede ser la conexión a una BD, un repositorio de almacenamiento, etc. Son clases desarrolladas en Java, por lo que se pueden emplear en toda la aplicación que la requiera.

Seguridad: es controlada por la máquina virtual de Java, que corrige la mayoría de problemas detectados de seguridad en los CGI, y ya no se aplican en los Servlets.

Portabilidad: se puede utilizar bajo cualquier plataforma de ejecución (S.O.) y en la mayoría de los servidores web como Apache, Microsoft IIS, WebStar. Están soportados directamente por plug-in.

Precio: normalmente todo el software utilizado con los Servlets es gratis, para el uso personal o sitio web de bajo nivel, a diferencia de Apache como servidor web gratuito, la mayoría de los servidores web comerciales son caros, cuando se tiene un servidor web comercial, se debe añadir en soporte para Servlets en alguno es gratuito y de muy bajo costo.

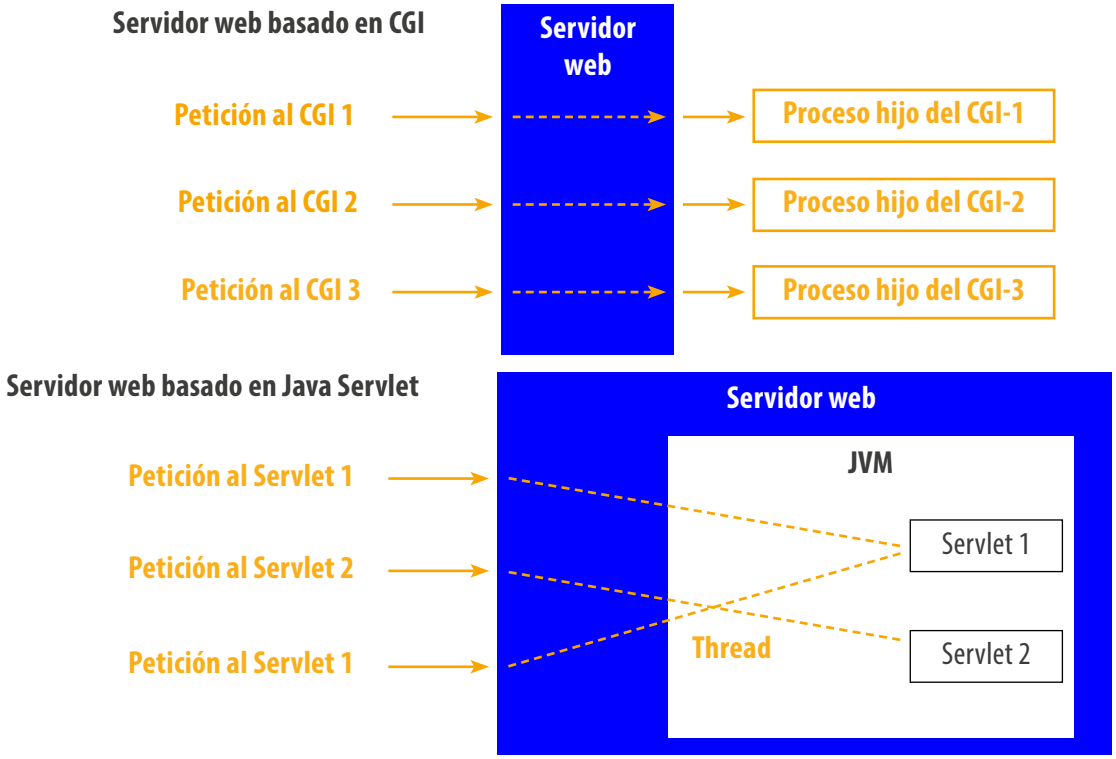


Figura 1. Funcionalidad de Servidor basado en CGI – Servlet
Fuente: Propia

JSP Java Server pages

Java JSP se aplica en las páginas web con la combinación del código HTML en un Java embebido de dos formas:

1. Scriptles: código Java-JSP multi-mensaje entre símbolos <% %>, donde cada instrucción debe ir separada por ";" punto y coma.
2. Expresiones: son variables que muestran un resultado, estas no finalizan con punto y coma, deben escribirse dentro de los símbolos <%= %>.
3. Declaraciones se insertan en la clase, no en un método <%! Declaraciones %>.
4. Comentarios JSP <%-- Comentario --%>.

Los elementos HTTP que componen los Servlets están predefinidos en los JSP, como:

- request: objeto HttpServletRequest que permite acceder a la información de la solicitud.
- response: objeto HttpServletResponse para generar la respuesta
- sesion: Objeto HttpSession asociado a la petición, Si no se define la sesión esta será null.
- application: el objeto ServletContext del contenedor web.
- out: Objeto JspWriter-Similar a un PrintWriter para ejecutar la salida para el usuario.

Puede ser llamados de forma directa, sin definirlos con anterioridad, veamos un ejemplo.

```
<%--
    Document   : index
    Created on : 13/08/2015, 01:21:04 PM
    Author     : gtabares
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Página JSP</title>
    </head>
    <body>
        <h1>Primer aplicación JSP</h1>
        La Hora actual es del sistemas es : <%= new java.util.Date()%>
    </body>
</html>
```

Imagen 1. Archivo html generado automáticamente.
Fuente: http://www.programandoconcafe.com/2011/03/Java-manejo-de-fechas-Javautildate.html

¿Cuáles son las Ventajas de JSP?

- Contra Active Server Pages (ASP). ASP es una tecnología propia y similar de Microsoft. Las ventajas de JSP están duplicadas. Primero, la parte dinámica está escrita en Java, no en Visual Basic, otro lenguaje específico de MS, por eso es mucho más poderosa y fácil de usar. Segundo, es portable a otros sistemas operativos y servidores Web.
- Contra los Servlets. JSP no nos da nada que no pudiéramos en principio hacer con un Servlet. Pero es mucho más conveniente escribir (y modificar) HTML normal que tener que hacer un billón de sentencias println que generen HTML. Además, separando el formato del contenido podemos poner diferentes personas en diferentes tareas: nuestros expertos en diseño de páginas Web pueden construir el HTML, dejando espacio para que nuestros programadores de Servlets inserten el contenido dinámico.
- Contra Server-Side Includes (SSI). SSI es una tecnología ampliamente soportada que incluye piezas definidas externamente dentro de una página Web estática. JSP es mejor porque nos permite usar Servlets en vez de un programa separado para generar las partes dinámicas. Además, SSI, realmente está diseñado para inclusiones sencillas, no para programas "reales" que usen formularios de datos, hagan conexiones a bases de datos, etc.

- Contra JavaScript. JavaScript puede generar HTML dinámicamente en el cliente. Este una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente. Con la excepción de las cookies, el HTTP y él envió de formularios no están disponibles con JavaScript. Y, como se ejecuta en el cliente, JavaScript no puede acceder a los recursos en el lado del servidor, como bases de datos, catálogos, información de precios, etc.

Para la implementación de los Servlet, se hace uso de las librerías básicas y de extensión para http:

- Javax.Servlet: entorno básico
- Javax.Servlet.http: extensión para Servlet http

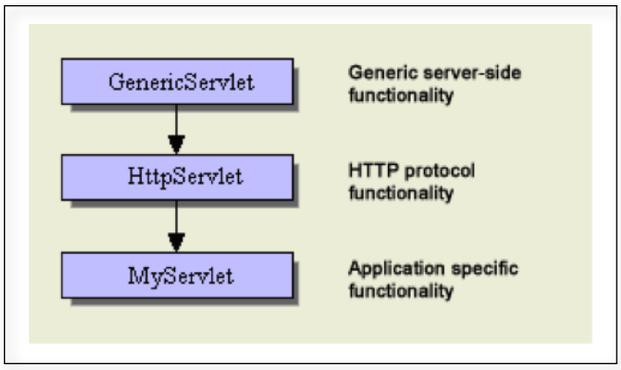


Imagen 2. Estructura del uso de Servlet básico
Fuente: http://ramannanda.blogspot.com.co/2009/06/Java-Servletsintroductioncompilingdeplo.html

Estructura básica de la definición de un Servlet:

```
import java.io.IOException;
import java.io.PrintWriter; // Para uso de entrada y salidas
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 *
 * @author gtabares
 */
@WebServlet(urlPatterns = {"/ModeloServlet"})
public class ModeloServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        /* El objeto "request" se utiliza para leer la cabecera HTTP, cookies,
        datos enviados por (GET o POST), El objeto "response"
        para indicar la respuesta*/
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            // out Se utiliza para enviar el contenido al cliente
        }
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Imagen 3. Servlet Java imagen de código generado por el autor dese IDE Netbeans
Fuente: Propia.

Ejemplo de la aplicación de un Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ModeloServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD "
                + "HTML 4.0 Transitional//EN\">"
                + "<html> <head><title>Hola Mundo</title></head>"
                + "<body> <h1>Hola Mundo</h1> </body></html>");
    }
}
```

Imagen 4. Método doGet del Servlet imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Normalmente es más práctico usar scriptlets que expresiones jsp.

- Muchas veces hay que comprobar valores, realizar acciones mucho más complejas, realizar un bloque de instrucciones cortas, etc.
- Por ejemplo, la siguiente expresión la podemos simplificar a través de un scriptles.

```
<p>Autor = <%= application.getInitParameter("Autor")%></p>

Scriptlet
<%
    String autor = application.getInitParameter("Autor");
    if ((autor == null) || (autor.trim().equals(""))) {
        autor = "Anónimo";
    }
    out.println("Autor = " + autor);
%>
```

Imagen 5. Scriptles en JSP imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

En el ejemplo Autor = <%= application.getInitParameter(“Autor”) %> Lo que trata es de ejecutar una instrucción directamente lo que venga a través de “autor”.

Con el Scriptles, se asigna sobre una variable de tipo String el parameter (Autor) y se valida a través de un condicional if si el autor esta en nulo y si al ejecutar el Scroptles no escribieron nada el algún formulario. Este se valida con autor.trim().equals(“”). En caso de que sea nulo o este en blanco se la asigna a la variable autor la etiqueta de Anónimo.

Declaraciones de variables en un scriptlets

Se pueden incluir declaraciones en la clase del Servlet con la instrucción <%- declaración %>.

- Este código se inserta fuera de los métodos de la clase, como nuevas declaraciones en la clase.
- Variables del Servlet.

  <%! private int edad; %>.
- Si se declaran variables con <% …. %> serán locales al scriptlet
- Métodos.

  Es mejor declararlos en una clase Java aparte.

  <%! private int contador = 0; %>

   Número de visitas que registra esta página <%= ++contador%>

Directivas

Las directivas se aplican a la clase Servlet generada, sintaxis de directiva.

<%@ directiva atributo = “valor” %>.

O bien

<%@ directiva atributo1 = “valor1” %>

atributo2=”valor2”

…

atributoN = “valorN” %>

Directivas comunes:

- Include: incluye otros ficheros que son de tipo jsp, pueden ser de tipo jsp, html o un Javascript.
- El fichero se referencia con una URL relativa a la página JSP o al servidor si va precedido de “/”.

  <%@ include file=”/URL” %>.
- Page Permite importar un paquete <% page import=”Java.util.*” %>

Ejemplo de Servlet con Netbeans

Servlet Básico en NetBeans.

Vamos iniciar con la creación de Servlet sencillo con el IDE de desarrollo de NetBeans:

Primera aplicación JSP

Primer paso: creamos un nuevo proyecto, File/New Project…

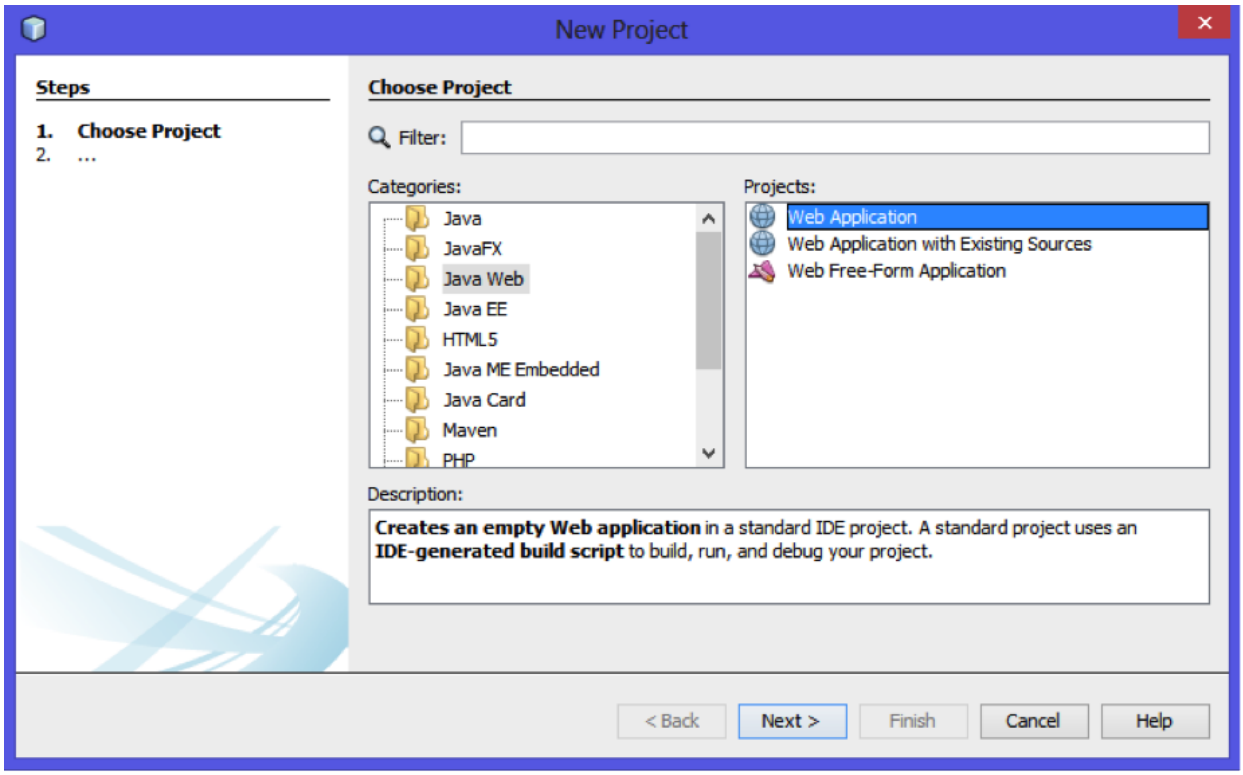


Imagen 6. Creando un nuevo proyecto imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

En la sección de categorías, seleccionamos de la lista Java Web y en la lista de proyectos seleccionamos la opción Web Application.

Clic en el botón Next (Siguiente), y le asignamos el nombre al proyecto: "JavaServlet".

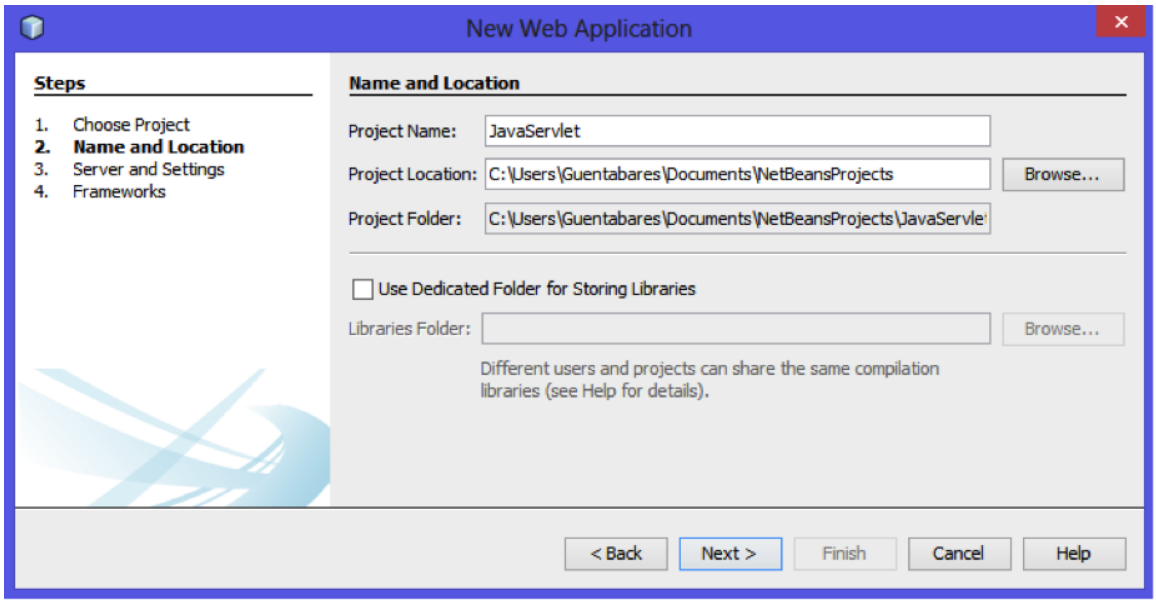


Imagen 7. Asignando nombre al proyecto JavaSerlvet imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Siguiente paso configurar el servidor web con el que realizaremos el ejercicio, es necesario para la ejecución del Servlet, ya que estos solo corren en un servidor web. Para este ejemplo utilizaremos el servidor que trae NetBeans por defecto llamado GlassFish.

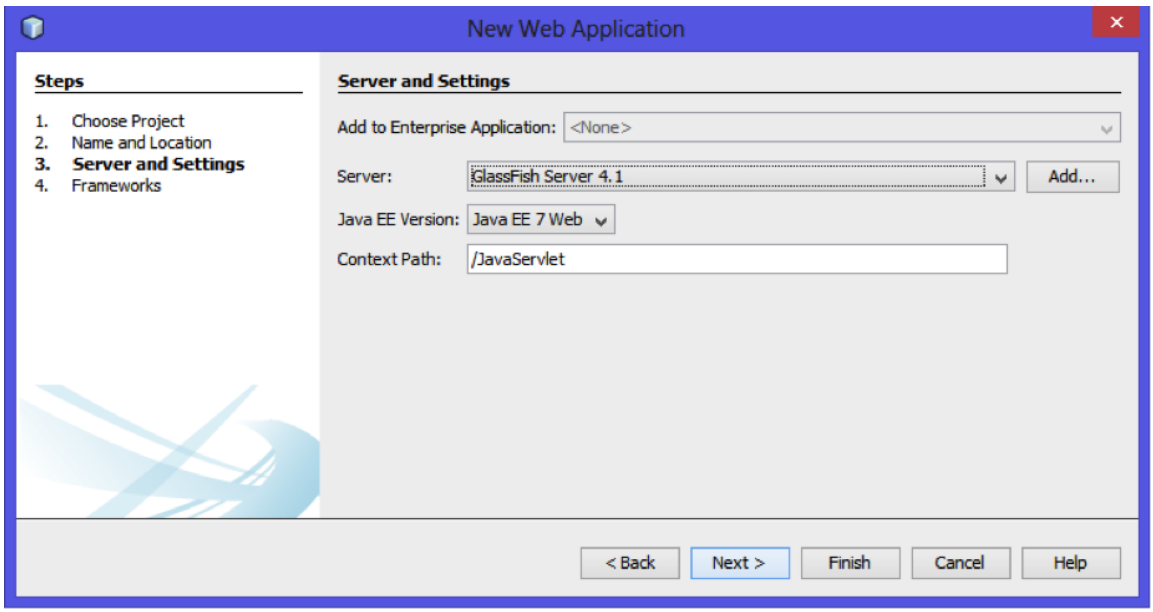


Imagen 8. Seleccionar el servidor GlassFish Server 4.1 imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Seleccionamos el botón Finish (Finalizar), y tendremos la siguiente estructura de nuestro proyecto.

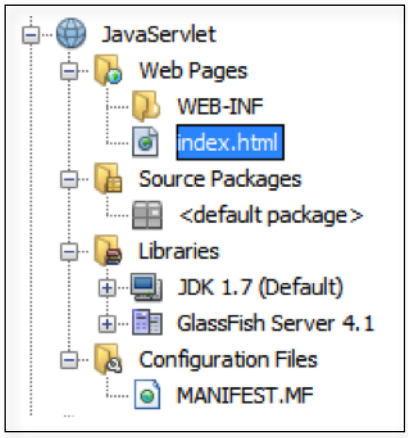


Imagen 9. Estructura de archivos del proyecto imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

En la estructura tenemos un archivo que se llama idex.jsp, en el cual podremos crear un formulario para llamar al archivo Servlet de la aplicación.

En la siguiente figura muestra el código del archivo index.jsp y en el escribiremos el código para crear un formulario. En la etiqueta <form action="Servlet" method="post">, se define el nombre del Servlet.

```
<!DOCTYPE html>
<!--
Author: Gtabares
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
        <form action="Servlet" method="post">
            Escribe tu nombre:
            <input type="text" name="nombre">
            <input type="submit" value="enviar">
        </form>
    </body>
</html>
```

Imagen 10. Formulario en HTML imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Vamos a crear el código, del Servlet del proyecto para esto damos clic derecho sobre el proyecto JaveServlet seleccionamos New / Servlet.

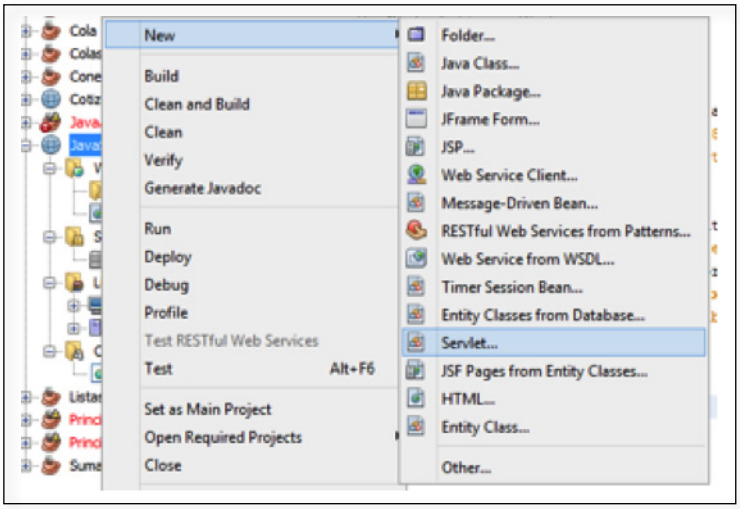


Imagen 11. Crear el Servlet de la aplicación imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

En la siguiente imagen se define el nombre del Servlet (para este ejemplo lo llamaremos Servlet, ya que fue el que definimos en la etiqueta del formulario), y pulsamos el botón clic, con esto hemos creado un archivo llamado Servlet.Java, el cual debemos de programar en Java.

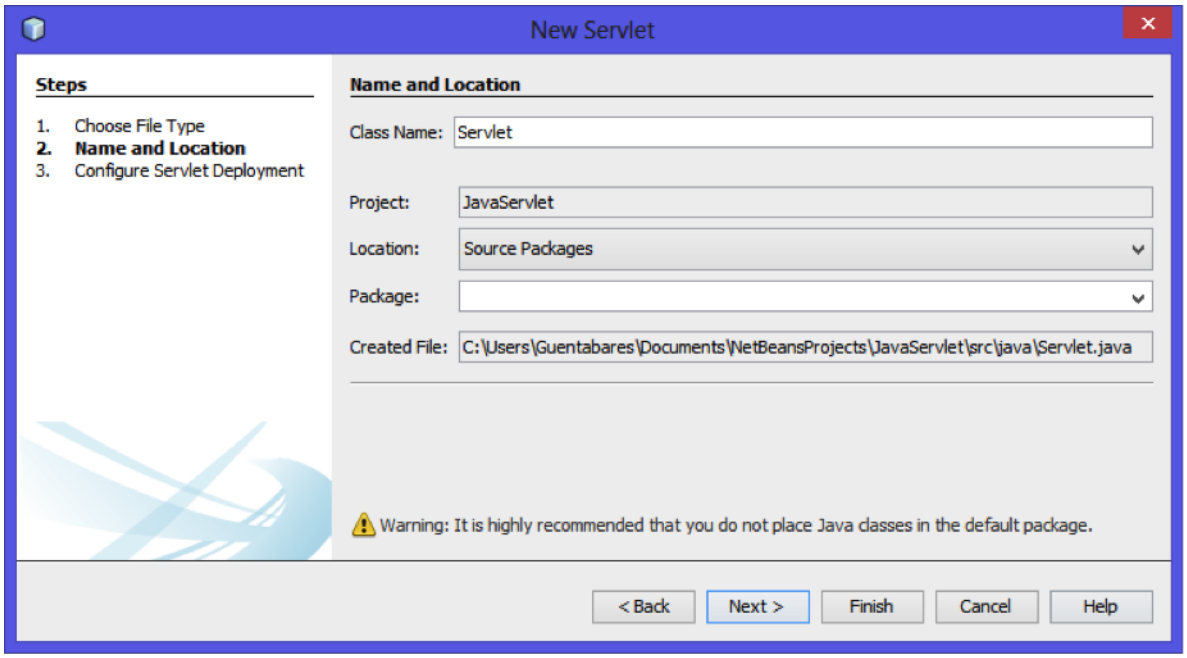


Imagen 12. Definir el nombre del Servlet imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Código generado del Servlet.Java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author Guentabares
 */
@WebServlet(urlPatterns = {"/Servlet"})
public class Servlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Tu nombre es: " + request.getParameter("nombre") + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Imagen 13. Personalizando el código del Servlet imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Modificamos y agregamos el siguiente código:

```
out.println("<h1>Tu nombre es: " + request.getParameter("nombre") + "</h1>");
```

Imagen 14
Fuente: Propia.

En el código anterior, hacemos uso de la instrucción request.getParameter("nombre") para obtener el nombre que se digita por el formulario en el archivo index..jsp.

Podemos ejecutar nuestro proyecto para verificar los resultados. En el botón run.

Situado en la parte superior del IDE.

Salida del proyecto en el explorar google Chrome.

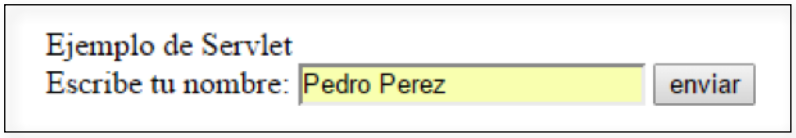


Imagen 15. Formulario de salida de la aplicación web imagen de código generado por el IDE Netbeans
Fuente: Propia.

Al digitar el nombre y pulsar sobre el botón enviar nos da como salida nuestro Servlet:

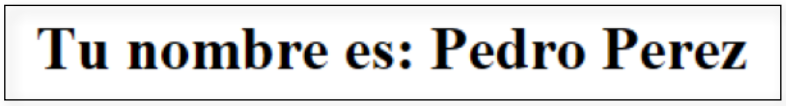
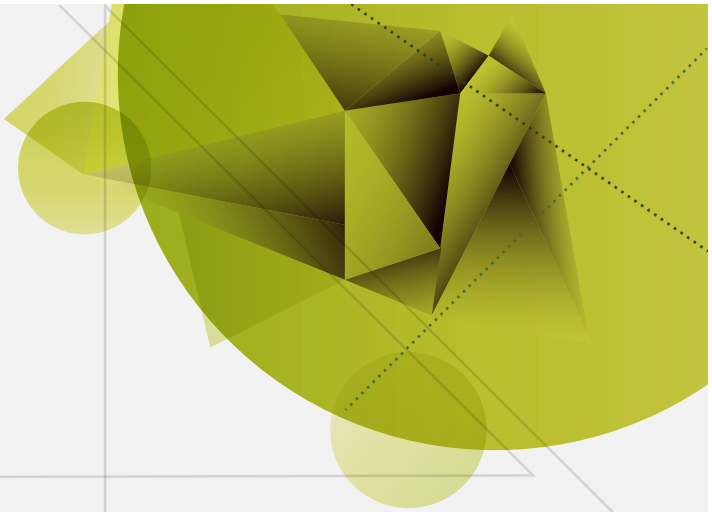


Imagen 16. Salida de la aplicación web. Imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

1

Unidad 1

Arquitectura Modelo-Vista-Controlador



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Un patrón de diseño en programación es un conjunto de buenas prácticas o un modelo estandarizado que permite ordenar la forma programación, es decir ordenar el código de alguna manera que nuestro proyecto se mantenga con el tiempo y facilitando a otros desarrolladores la comprensión de forma fácil del código y estructura del proyecto.

U1

Metodología

El desarrollo de esta cartilla este desarrollada bajo en tema de patrones de diseño MVC, se recomienda al estudiante seguir de forma adecuada el desarrollo del tema, apoyándose en el material complementario. Lecturas de referencias, Videos capsulas. Es importante estar muy activo en la video conferencia es un espacio donde se pueden aclarar muchas dudas que se presenten con el tema.

En esta semana se realizara la primera evaluación a través de un Quiz, enfocado a los conceptos del tema de la semana 2. Adicional hay una actividad de repaso que le permitirá al estudiante poder aplicar los conocimientos adquiridos hasta el momento.

Arquitectura Modelo-Vista-Controlador

El patrón de desarrollo Modelo-Vista-Controlador establece unos lineamientos de buenas prácticas de programación, según la cual todo programa orientado a objetos bajo este patrón consta de tres partes: el Modelo, La Vista y el Controlador.

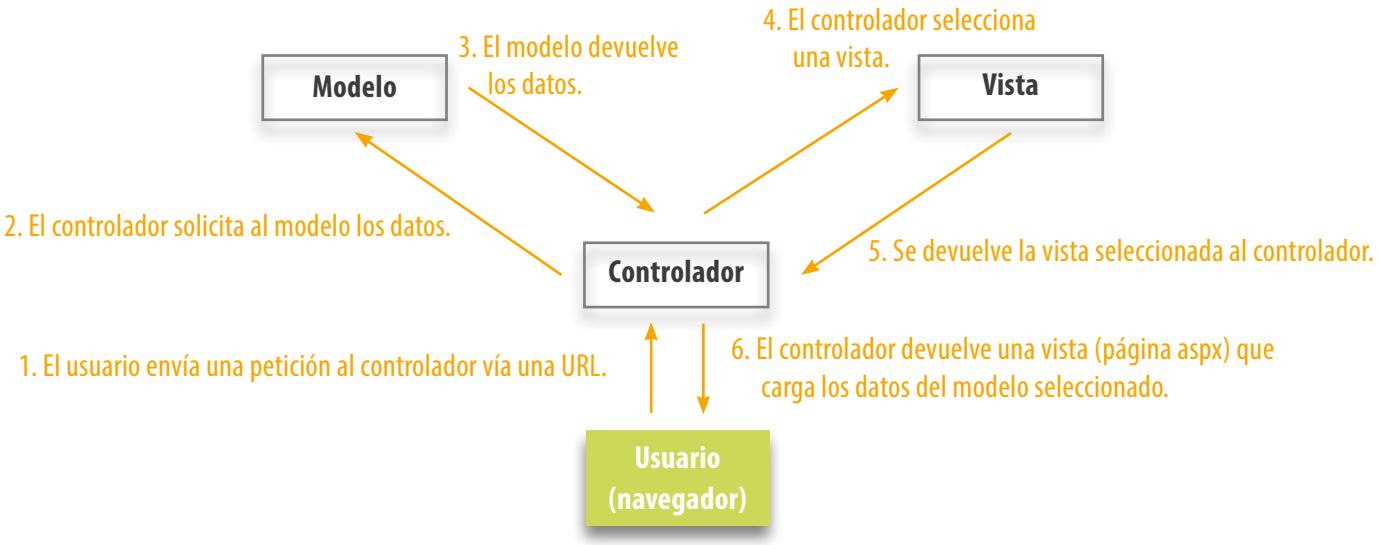


Figura 1: Patrón de desarrollo Modelo-Vista-Controlador
Fuente: Propia.

Esta arquitectura pretende que el programador separe la parte lógica del programa, de la parte de presentación. Como siempre, al dividir un problema, se disminuye la complejidad de su solución, y permite que más tarde el mantenimiento sea mucho menos costoso en términos de tiempo y dificultad, ya que dependiendo de la naturaleza (del modelo, de la vista o del control) del aspecto sobre el que se requiera un cambio o un ajuste, su ubicación es más rápida y los efectos de efectuar sobre ella el mantenimiento serán más claros y predecibles. Si por ejemplo lo que se quiere es cambiar un logo en una pantalla, pues este cambio solo afecta una parte de la vista, sin alterar al modelo o al controlador. Si por el contrario se requiere un cambio en un método de un objeto como por ejemplo el cambio en el cálculo

de retención en la fuente, solo se tendrá que modificar el método dentro de la clase correspondiente al objeto dentro del modelo.

La aplicación del patrón MVC facilita también la reutilización de partes del programa. Si por ejemplo una interface igual a una ya programada es requerida, pues se puede utilizar sin problema aunque la lógica que la vaya a utilizar sea diferente.

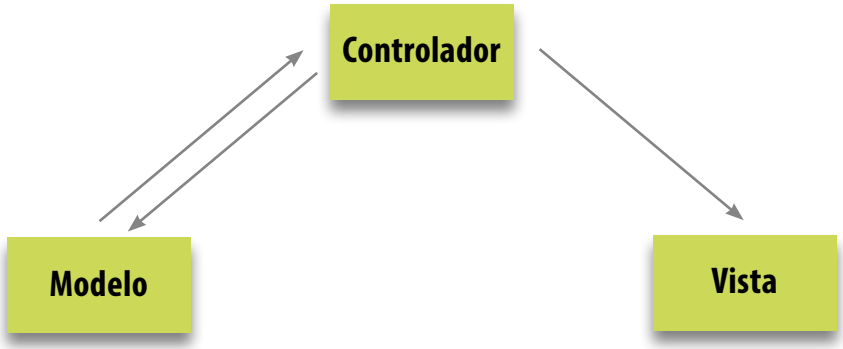


Figura 2. MVC Simplificado imagen tomada del sitio
Fuente: http://juandarodriguez.es/cursosf20/unidad2.html

Modelo

El Modelo es la parte del programa que contiene las clases correspondientes al modelo de datos y es el encargado de interactuar con la base de datos.

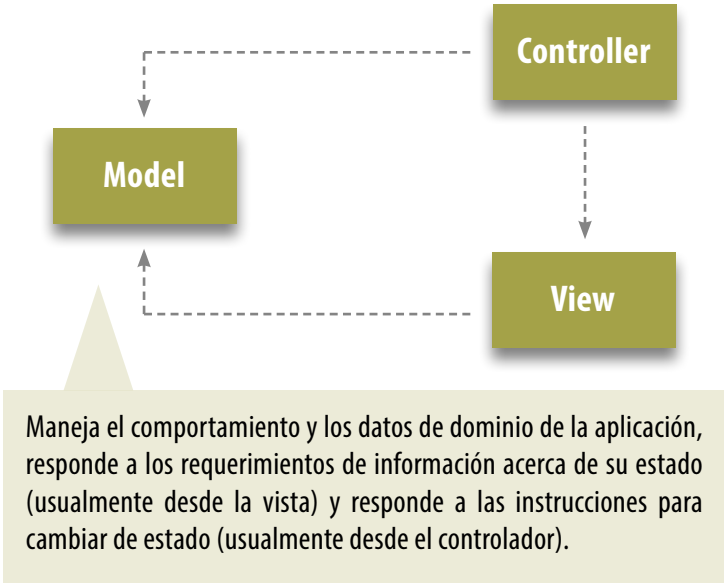


Figura 3. Descripción del Modelo imagen tomada del sitio
Fuente: Propia.

Vista

La Vista es la encargada de las interfaces con las cuales el programa se relaciona con el usuario, permitiendo las entradas y las salidas de datos, es la interfaz visible a los operadores y/o a otros sistemas.

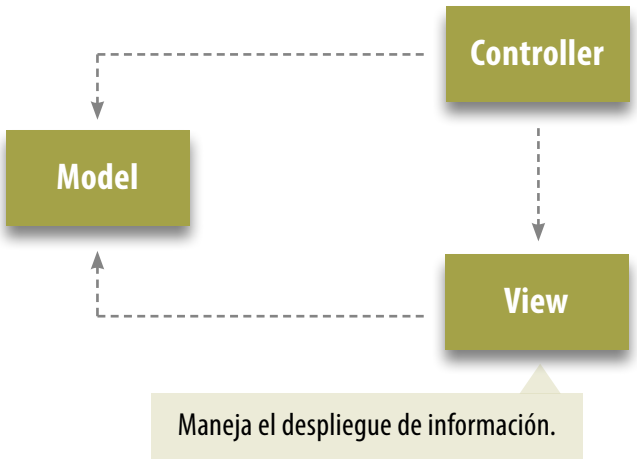


Figura 4. Descripción del Vista
Fuente: Propia.

Controlador

Es el encargado del bloque de control, su función es recibir y/o enviar datos al a la interfaz y además crear instancias de los objetos del modelo con el objetivo de poder ejecutar sus métodos dependiendo de la funcionalidad del programa.

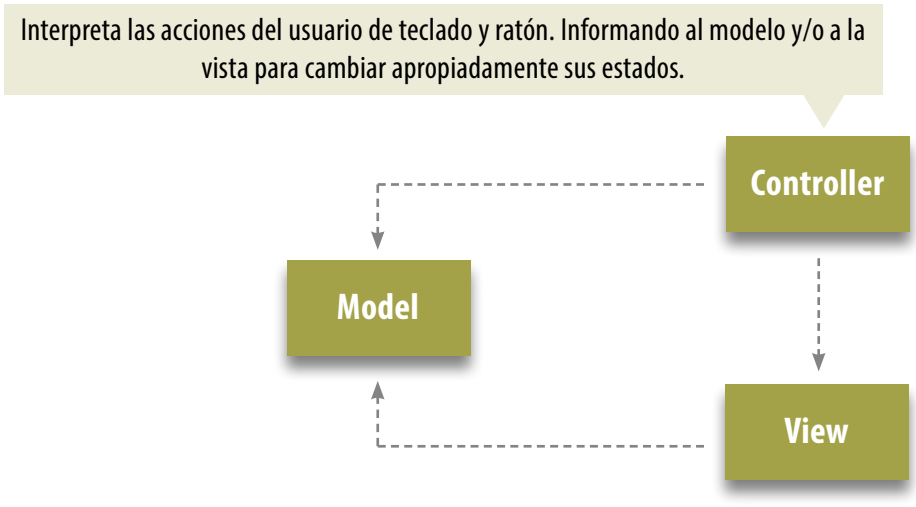


Figura 5 Descripción del Controlador
Fuente: Propia.

Ejemplo

Escenario: desarrollar un programa para simular una calculadora que pueda sumar, restar, multiplicar y dividir dos operandos que un usuario digita. Se deben mostrar los operandos, la operación y el resultado.

Un ejemplo sencillo en el que se aplicaran cada uno de los conceptos tratados hasta el momento, es decir desde el planteamiento del escenario hasta su arquitectura MVC.

Escenario

Identificación de objetos en el escenario.

Para identificar los objetos se seguirá el procedimiento de elaborar una lista de los **sustantivos** que están mencionados en el texto del problema:

- Calculadora
- Operando
- Operación
- Usuario
- Resultado

Después de esta operación analizamos cada uno de ellos, para encontrar aquellos que dentro del escenario existen independientemente.

Calculadora tiene existencia propia, no depende de otro objeto, luego es válida su inclusión como objeto.

Operando es un **dato** que será utilizado por la **calculadora** para efectuar alguna operación, por sí solo **no** tiene sentido su existencia, luego no es un objeto. En este caso aparecen 2.

Operación solo tiene existencia dentro de la **calculadora**, esta es quien realiza la **operación,** de lo cual se deduce que No es un **objeto.**

Usuario, es quien utiliza el programa, luego aunque se menciona dentro del texto, está por fuera del escenario, luego no es un **objeto**.

En conclusión este escenario **únicamente** tiene al objeto **calculadora**.

Abstracción de objetos

Como el único objeto identificado es calculadora, solo se efectuara la abstracción de él, colocándole un nombre e identificando sus componentes (o atributos o características), lo mismo que los métodos que permiten que los valores de esos componentes puedan ser modificados.

Los componentes normalmente son los sustantivos que hemos rechazado como objetos, pero que están dentro del escenario, es decir que forman parte de la solución del problema:

- Operando1
- Operando2
- Operación resultado

Teniendo la lista de componentes, podemos elaborar una lista de metodos, analizando cada componente y deduciendo que modificaciones puede tener el valor del componente y como se pueden efectuar:

Para **operando1** se requiere que el valor sea uno de los que interviene en la operación, luego el cambio de valor se obtiene "poniendo" dentro del valor correspondiente. Debemos tener un Método que permita hacer esto:

Establecer_operando1 ()

Como el resultado pide que "se muestren los operandos", hay que tener otro método que "extraiga" el valor del OPERANDO1:

Obtener_operando1 ()

Con operando2 sucede lo mismo, luego también se requieren dos métodos:

Establecer_operando2 ()

Obtener_operando2 ()

Existen cuatro posibles valores para el componente operación: "suma", "resta", "multiplicación", "división". El componente como es normal, solo tendrá uno de ellos y hay que "ponerlo" dentro de él. Para este efecto se utilizará el método:

Establecer_operación ()

Al final también hay que mostrar el valor de operación, luego hay que extraer el valor de OPERACIÓN para mostrarlo:

Extraer_operación ()

Como su nombre lo indica, el componente resultado no es "cargado" con un valor específico predeterminado (como sí lo son los operandos y la operación), sino que resulta de efectuar la operación correspondiente, luego se necesita un método que defina qué operación se debe efectuar y que la realice:

Calcular_resultado () Y para "extraerlo"

posteriormente:

Obtener_resultado ()

En este punto ya tenemos todos los elementos que conforman el objeto calculadora del problema y lo podemos diagramar:

Abstracción del objeto calculadora

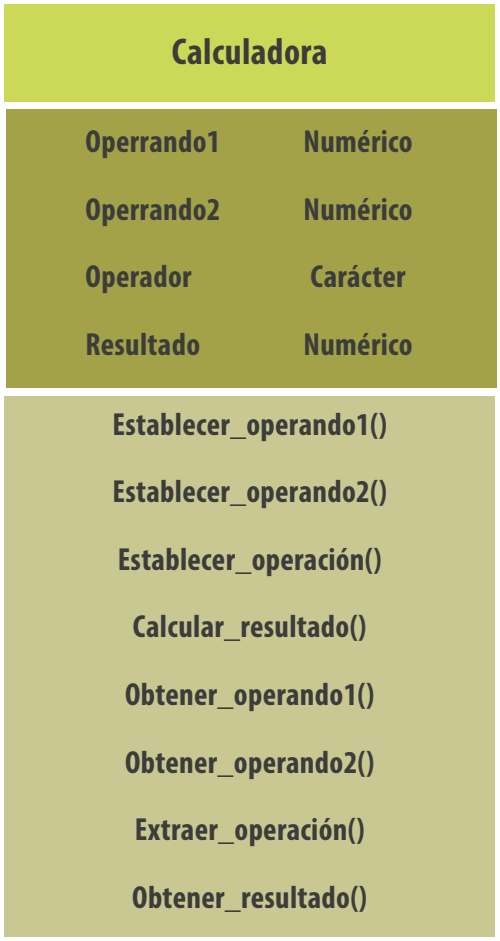


Figura 6. Abstracción de la clase
Fuente: Propia.

En el diagrama para cada componente se debe especificar el tipo de datos que recibe. Este diagrama es el mismo diagrama de clase del objeto calculadora.

Diagrama de clases del problema

Usando la arquitectura MVC, se debe especificar el modelo y el control, para este caso la vista estará incluida dentro del control.

Como solo hay un objeto, entonces el número de clases que conforman el modelo es solo una:

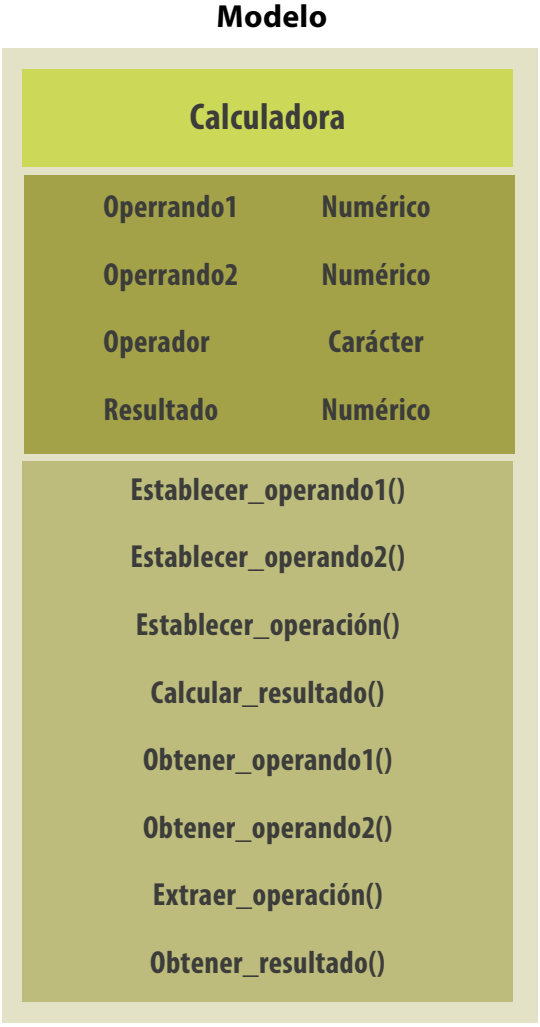


Figura 7. Modelo lógico Bean
Fuente: Propia.

Bloque modelo del diagrama de clases del problema

Necesitamos el bloque de control, el cual constara únicamente de la clase principal que debe tener siempre todo controlador. A esta clase la llamaremos Calculadora_basica () y el diagrama completo quedara de la siguiente forma:

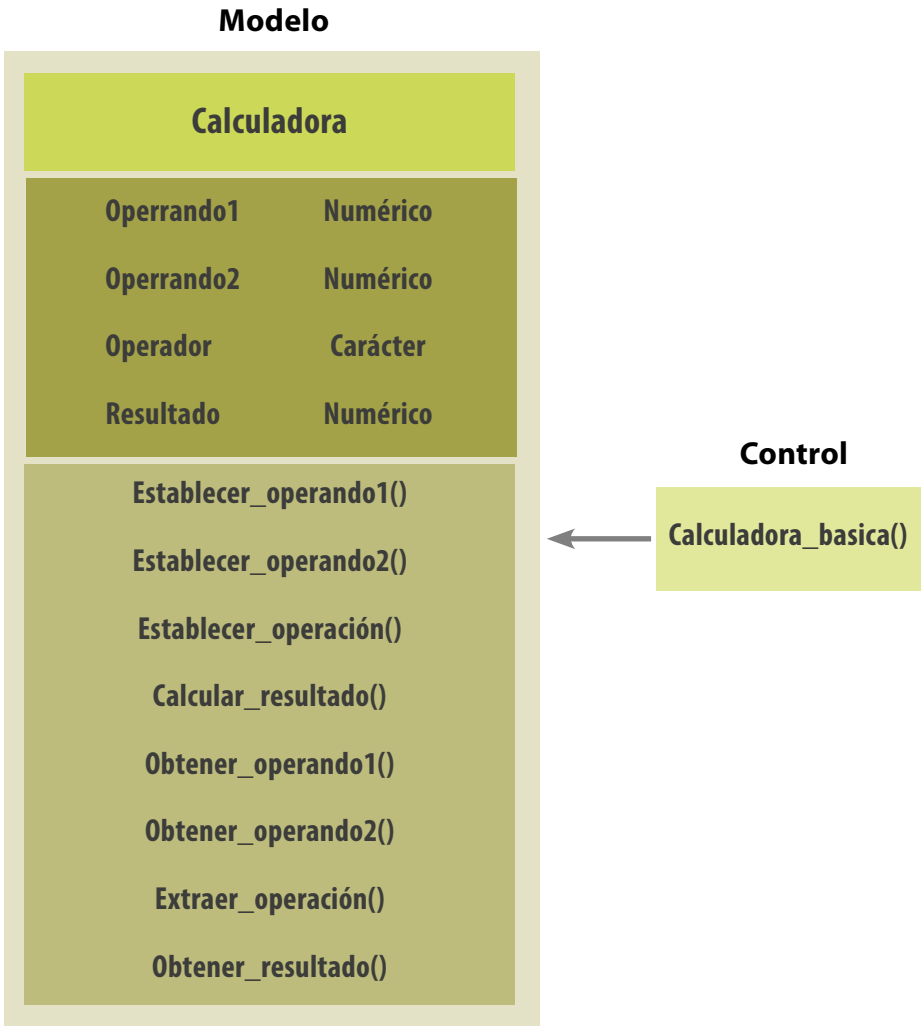


Figura 8. Diagrama de clases del problema
Fuente: Propia.

Diagrama de clases del problema

Desarrollo Web: MVC. Patrones de diseño web.

Una calculadora básica como aplicación, Queremos sumar dos números.

Inicialmente vamos a separar las cosas que hace el servidor y las cosas que van de la parte del cliente.

1. El cliente hace una petición desde la página principal (index.jsp) de la app Calculadora.
2. El servidor procesa la pantalla inicial, que permite ingresar dos números.
3. El cliente suministra dos números y presiona el botón Sumar del formulario.

4. Esto genera que se haga una solicitud a un pedido a otra página, que debe tomar los valores ingresados por el cliente (a través de los métodos get o post), efectuar la suma y mostrar ese resultado por pantalla.

   El formulario de index.jsp dice:

```
<form method="post" action="calcular">
```

¿Qué se puede colocar como parámetro en la opción action de un form?

- links externos («http://....»)
- links internos dentro del mismo proyecto
  - páginas html
  - otros recursos (archivos pdf, xls, con un formato que el browser pueda mostrar)
  - o se puede redireccionar al destino a una clase Java que se ejecute del lado del servidor y devuelve una página html al cliente. Tenemos dos posibilidades:
    - páginas JSP
    - servlets

El method hace post/get a un action «calcular». ¿Dónde va a parar ese pedido entonces?

Abrimos el archivo web.xml de src/main/webapp/WEB-INF y vemos que es un xml que mapea nombres con .... servlets.

```
<servlet>
    <servlet-name>calcular</servlet-name>
    <servlet-class>ar.edu.unq.tpi.labso.calculadora.control.CalcularServlet</servlet-class>
</servlet>
```

El código anterior permite relacionar un nombre («calcular») con un package name + nombre de la clase del servlet.

Podemos probar en el archivo web.xml cambiando el contenido de la url-pattern y el package y analizar los errores que arroja.

Un Servlet

Es una clase Java que extiende HttpServlet. CalcularServlet en este ejemplo implementa un único método doPost. El servlet está pensado para recibir un pedido (http request), procesarlo y devolver código html que pueda ser formateado por el browser.

Métodos son importantes para un servlet

- doPost(), cuando recibimos un http request vía post.
- doGet(), ídem pero vía get.

Estado de un servlet

Si un servlet es una clase Java, ¿puede tener estado?

- Técnicamente e Java, los servlets son singletons, entonces por más usuarios que haya conectados al web server, sólo se tiene una instancia de cada servlet de la aplicación.
- Los atributos o variable que se han definido en un servlet, estas son compartidas entre todas las sesiones activas, el acceso a esas variables es concurrente.
- entonces el manejo del estado de una página no es trivial:
  - El servidor no retiene información por sesión de usuario.
  - y encima al pasar entre página y página pasó por el cliente que está limitado por lo que el HTML me deja hacer.

MVC en Web

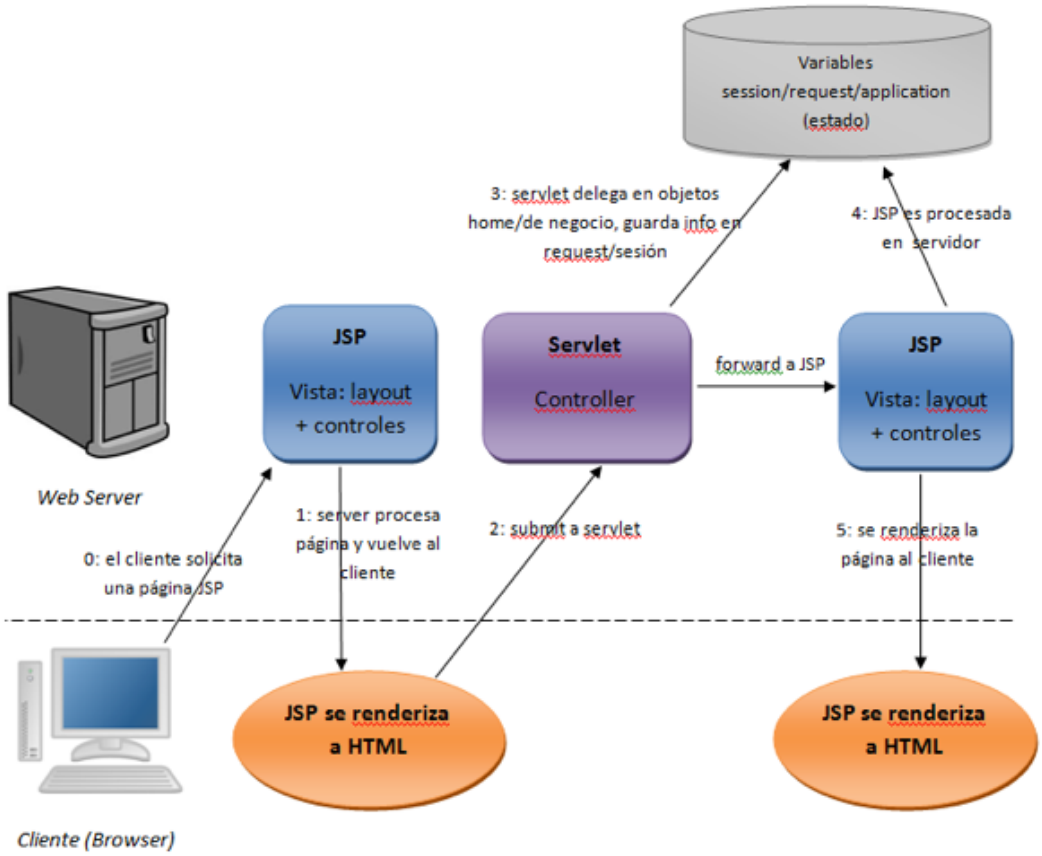


Figura 9. Diagrama del patrón MVC para aplicaciones web. Imagen tomada del sitio http://algo3.uqbar-project.org/temario/03-intro-web/java/web-mmvc

A través del siguiente ejemplo implementaremos un ejercicio para aplicar el patrón de desarrollo MVC en aplicaciones web.

Crearemos los siguientes archivos que describen lo siguiente:

Archivo Entradas.jsp página (vista).

Archivo User.Java Bean (Modelo).

Archivo Procesos.Java Servlet (Controlador).

Archivo Resultado.jsp página (vista).

Creamos una aplicación Java Web / Web Application, con el nombre de implementación-MVC, definimos el servidor GlassFish Server y clic en finish.

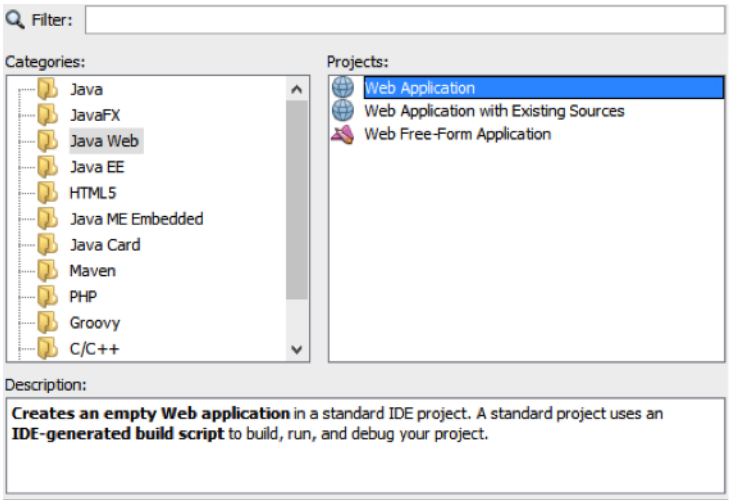


Figura 10. Creación aplicación web imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Ya tenemos el proyecto creado, con la estructura de archivos generado y el index.html que lo podemos eliminar para crear el jsp.

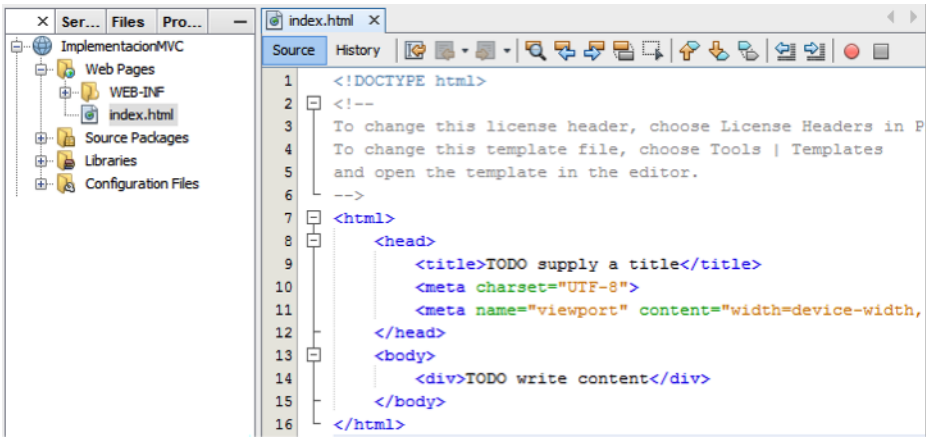


Figura 11. Estructura de archivo proyecto Web Application imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Para crear el primer archivo entrada.jsp correspondiente a la vista, da clic derecho del mouse sobre Web Pages, new, JSP…

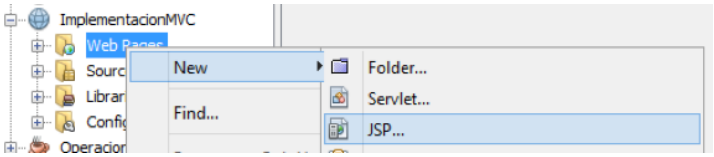


Figura 12. Creando el archivo JSP imagen generada por el autor desde IDE Netbeans
Fuente: Propia.

Creamos el siguiente código en el archivo Entrada.jsp para el formulario de entrada de los datos del usuario.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Información del Usuario</title>
    </head>
    <body>
        <h1>Información del Usuario</h1>
        <form method="post" action="">
            Nombre: <input type="text" name="priNombre"/><br>
            Apellido: <input type="text" name="priApellido"/><br>
            Email: <input type="text" name="email"/><br>
            <input type="submit" value="enviar"/>
        </form>
    </body>
</html>
```

Figura 13. Código archivo entrada.jsp imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Para crear el siguiente archivo clic derecho sobre Source Packages, seleccionamos new, Java class con el nombre de User.Java, y creamos el paquete com.ejer.modelo.

En el archivo User.Java es importante crear el constructor de la clase con los argumentos definidos de la clase, ejemplo.

```
public User(String priNombre, String priApellido, String email){
    this.priNombre = priNombre;
    this.priApellido = priApellido;
    this.email = email;
}
```

Figura 14. Definición del constructor de la clase User.Java imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

A continuación creamos los métodos Getter y Setter de la clase en el archivo User.Java como se muestra a continuación.

```java
package com.ejer.modelo;

/**
 * @author gtabares
 */
public class User {
    private String priNombre;
    private String priApellido;
    private String email;

    public User(String priNombre, String priApellido, String email){
        this.priNombre = priNombre;
        this.priApellido = priApellido;
        this.email = email;
    }
    /**
     * @return the priNombre
     */
    public String getPriNombre() {
        return priNombre;
    }

    /**
     * @param priNombre the priNombre to set
     */
    public void setPriNombre(String priNombre) {
        this.priNombre = priNombre;
    }

    /**
     * @return the priApellido
     */
    public String getPriApellido() {
        return priApellido;
    }

    /**
     * @param priApellido the priApellido to set
     */
    public void setPriApellido(String priApellido) {
        this.priApellido = priApellido;
    }

    /**
     * @return the email
     */
    public String getEmail() {
        return email;
    }

    /**
     * @param email the email to set
     */
    public void setEmail(String email) {
        this.email = email;
    }

}
```

Figura 15. Código clase Bean User.Java imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Para crear el siguiente archivo clic derecho sobre Source Packages, seleccionamos new, Java class con el nombre de Proceso.Java, y creamos el paquete com.ejer.controlador, y creamos el siguiente código:

```
package com.ejer.controlador;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ejer.modelo.User;
import javax.servlet.RequestDispatcher;

/**
 *
 * @author gtabares
 */
public class Procesos extends HttpServlet{
    public void doPost(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException{
        String priNombre = request.getParameter("priNombre");
        String priApellido = request.getParameter("priApellido");
        String email = request.getParameter("email");

        User user = new User();
        user.setPriNombre(priNombre);
        user.setPriApellido(priApellido);
        user.setEmail(email);

        request.setAttribute("user", user);

        RequestDispatcher dispatcher = request.getRequestDispatcher("resultado.jsp");
        dispatcher.forward(request, response);
    }
}
```

Figura 16. Código controlador clase Procesos.Java imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Editamos el archivo web.xml, si este no está en la estructura del proyecto, lo creamos con clic derecho sobre el proyecto, new, other y buscamos en web el tipo de archivo Standard Deployment Descriptor (web.xml).

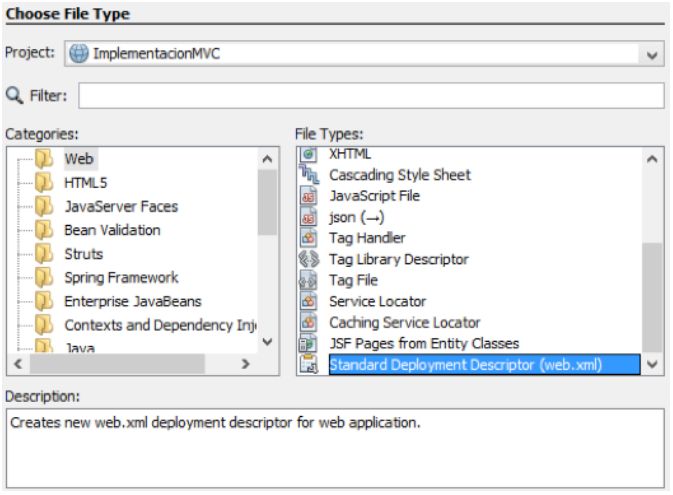


Figura 17. Creando el archivo web.xml imagen generada por el autor desde IDE Netbeans
Fuente: Propia.

Adicionamos el siguiente código:

```
ersion="1.0" encoding="UTF-8"?>

p xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">

rvlet>
 <servlet-name>Procesos</servlet-name>
 <servlet-class>com.ejer.controlador.Procesos</servlet-class>
ervlet>

rvlet-mapping>
 <servlet-name>Procesos</servlet-name>
 <url-pattern>/Procesos</url-pattern>
ervlet-mapping>

ssion-config>
 <session-timeout>
     30
 </session-timeout>
ession-config>
lcome-file-list>
 <welcome-file>entrada.jsp</welcome-file>
```

Figura 18. Código xml del archivo web.xml imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Creamos un nuevo archivo Resultado.jsp para mostrar al usuario la información en la vista resultado.

```
<%--
    Document   : Resultado
    Author     : gtabares
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="com.ejer.modelo.User" %>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Resultado</title>
    </head>
    <body>
        <h1>Información del Usuario</h1>
        <%
            User user=(User) request.getAttribute("user");
        %>
        <%=user.getPriNombre() %><br>
        <%=user.getPriApellido()%><br>
        <%=user.getEmail()%><br>

    </body>
</html>
```

Figura 19. Código Archivo Resultado.jsp imagen de código generado por el autor desde IDE Netbeans
Fuente: Propia.

Ejecutamos la aplicación y muestra el formulario de ingreso de datos

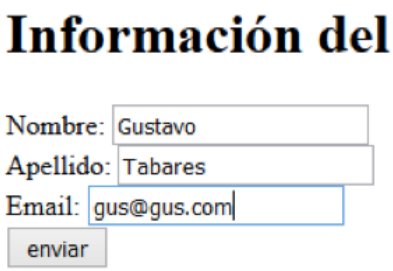


Figura 20. Formulario de datos imagen generada por el autor desde IDE Netbeans
Fuente: Propia.

Clic en botón enviar, el controlador procesa los datos y los muestra en la vista Resultado.jsp

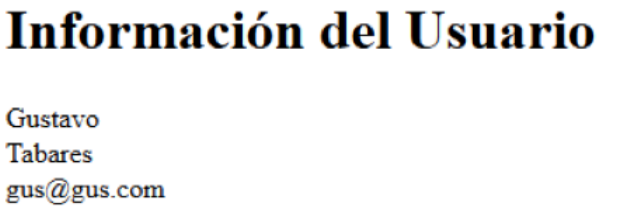
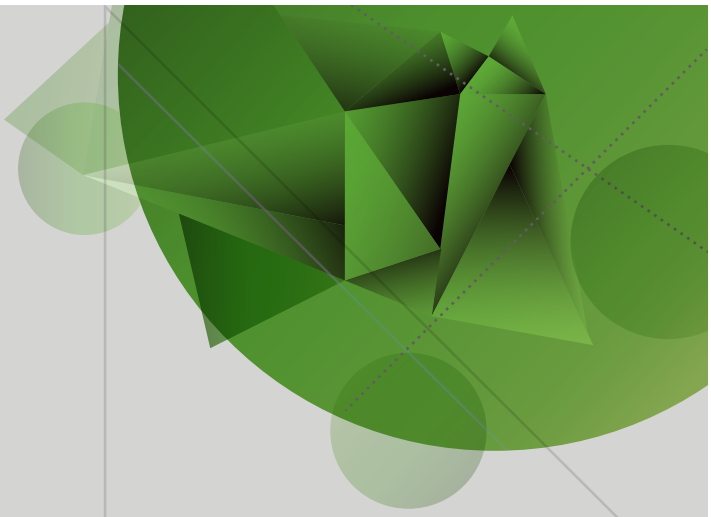


Figura 21: Datos de respuesta imagen generada por el autor desde IDE Netbeans
Fuente: Propia.

2

Unidad 2

Java Sever Faces Tecnología



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El uso de plantillas para el desarrollo de aplicaciones web, nos permiten cumplir con una estructura de desarrollo definida de forma estándar y cumplir con la calidad que se requiere a la hora de programar, JSF ofrece una variedad de sentencias que aplicadas correctamente se puede lograr construir aplicaciones exitosas.

U2

Metodología

El desarrollo de esta cartilla está desarrollada bajo en tema referentes a JSF, se recomienda al estudiante seguir de forma adecuada el desarrollo del tema, apoyándose en el material complementario. Lecturas de referencias, Videos capsulas. Es importante estar muy activo en la video conferencia, es un espacio donde se pueden aclarar muchas dudas que se presenten con el tema.

En esta semana se realizara la primera evaluación a través de un quiz, enfocado a los conceptos del tema de la semana 3. Adicional hay una actividad de repaso que le permitirá al estudiante poder aplicar los conocimientos adquiridos hasta el momento.

U2

Desarrollo temático

Java Sever Faces Tecnología

La tecnología JavaServer Faces, es un componente de trabajo integrado por interfaces de usuario que están del lado del servidor permitiendo la generación de aplicaciones web con tecnología Java.

Los principales componentes de la tecnología JSF

La implementación de una API y una referencia: para generar componentes UI y controlar su estado; el control de eventos, validaciones del lado del servidor y conversión de datos; permitiendo la navegación entre páginas, soportando los estándares de internacionalización y accesibilidad; proporcionando extensibilidad para todas estas características.

Esta implementación de tecnología facilita la construcción de aplicaciones web con UIs del lado del servidor, con una programación bien definida y las librerías necesarias para su correcta implementación. Con un mínimo esfuerzo de programación, podemos lograr:

- Conectar los eventos que se generan del lado del cliente al código de la aplicación del lado del servidor.
- Mapear los componentes UI a una página de datos del lado del servidor.
- La construcción de Interfaces de usuarios UI con componentes reutilizables y extensibles.
- Una tecnología que se ejecuta del lado del servidor y no del lado del cliente.
- La interfaz de usuario es tratada como un conjunto de componentes UI.

Características principales

- Generar las vistas a través de JSF, incluyendo una biblioteca de etiquetas propias para los elementos de los formularios.
- Desde el código HTML, asocia a las vista con formularios, un conjunto de objetos Java manejados por el controlador (Beans) que permiten la captura, manipulación y visualización de los datos mostrados en los diferentes elementos de los formularios.
- Proporciona una serie de etapas en el procesamiento de la petición, como la validación, reconstrucción de las vistas, recuperaciones de los datos de los elementos.

- Utiliza para el controlador un fichero de configuración con el formato XML.
- Permite la creación de nuevos elementos de la interfaz o modificar los ya existentes.
- Hace parte del estándar J2EE, lo que permite muchas posibilidades de la creación de la capa de presentación logrando el control de una aplicación web Java, como Struts y otros frameworks.

Aplicación Java Server Faces

Los pasos para realizar una aplicación Java Server Faces son los siguientes:

- Extracción de los objetos del modelo lógico, que contendrán los datos necesarios de la aplicación.
- Añadir las declaraciones del Java-bean al controlador del fichero de configuración de la aplicación.

La siguiente grafica muestra los archivos que hacen parte de una aplicación Java Server Faces.

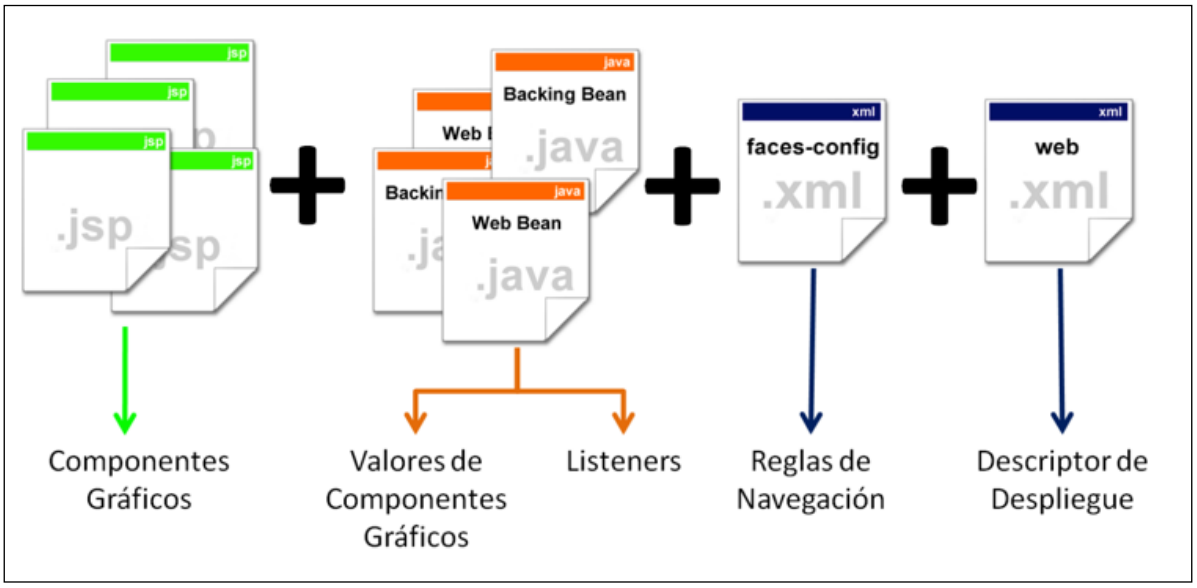


Figura 1. Conjunto de aplicaciones que hacen parte de una aplicación JSF
Fuente: Propia.

Aplicación Java Server Faces

Las etiquetas usadas en JSF

JSF, contiene una colección de básica de etiquetas disponibles que permiten crear de forma fácil, páginas con los componentes dinámicos de una página web. A continuación veremos algunas:

- h:inputText. Está incluye un campo de texto normal para la entrada de datos.
- h:inputHidden. Está etiqueta incluye un campo de entrada oculto del formulario.
- h:commandButton. Las acciones que se generan en los formularios son asociadas a los botones.
- h:inputSecret . Campo de tipo contraseña editable.
- h:form. Etiqueta Formulario JSF.
- h:outputLabel. Etiqueta para mostrar etiquetas de tipo texto.
- h:outputText. Etiqueta para salida de tipo texto.
- h:selectOneRadio. Permite crear botones de selección única.
- h:inputTextarea. Permite crear un campo de texto multilínea.
- h:outputLabel. Muestra un texto de salida fijo.

Iniciaremos el proceso para crear una aplicación web con tecnología JavaServer Faces, es un proceso fácil y amigable, para desarrollar una aplicación se requiere de las siguientes tareas:

- Desarrollar beans mejorados.
- Agregar declaraciones de manejo de bean.
- Crear páginas web utilizando etiquetas.
- Mapear la instancia FacesServlet.

Durante esta semana describiremos estas tareas, creando una aplicación de ejemplo en JavaServer Faces.

Inicialmente crearemos una aplicación Hello que contendrá un backing bean y una página web, que le permitirá al cliente cuando ingrese y ejecute la aplicación, imprimir un mensaje de HelloWord.

Creando el bean de respaldo (backing bean)

En la tecnología JSF el componente backing bean, es un complemento de JavaBean, Los componentes se asocian a una página con un backing bean, donde se provee la lógica de la aplicación. El siguiente código del ejemplo Hello.Java muestra un ejemplo aplicable de backing bean.

```
package hello;
import Javax.faces.bean.ManagedBean;
@ManagedBean
public class Hello {
  final String world = "Hello World!";
  public String getworld() {
    return world;
  }
}
```

Analicemos un poco, el backing bean asigna un valor a la variable mundo con la cadena "Hola Mundo" la anotación @ManagedBean se encarga de registrar el backing bean como un recurso de la implementación JSF.

Creando la página Web

En una aplicación JSF, las páginas son creadas en XHTML, En el siguiente ejemplo de página web XHTML, tiene el siguiente código.

```
package hello;
import Javax.faces.bean.ManagedBean;
@ManagedBean
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://Java.sun.com/jsf/html">
 <h:head>
   <title>Facelets Hello World</title>
 </h:head>
 <h:body>
   #{hello.world}
 </h:body>
</html>
```

El lenguaje de expresiones EL (Expresision Language), conecta la página web con el backing bean, la expresión con el valor #{hello.world}, es el encargado de recuperar el valor de propiedad world del backing bean Hello. Se usa la referencia hello al backing bean Hello. Si no se especifica nombre en la anotación @ManagedBean, Se puede llamar al backing bean con el mismo nombre y la primera letra de la clase en minúsculas.

Mapping De Faces Servlet Instancia

El mapeo de Faces Servlet, se realiza modificando en deployment descriptor web.xml, un mapeo típico del servlet Faces Servlet es como sigue:

```
<servlet>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>Javax.faces.webapp.FacesServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

El segmento de archivo precedente, representa parte de un archivo de descripción de despliegue. El archivo de despliegue puede contener más información de configuración de la aplicación con JavaServer Faces.

El mapeo de FacesServlet se realiza de forma automática cuando se usa el IDE de desarrollo Netbeans.

El ciclo de vida de la aplicación Hello

Las aplicaciones web tienen un ciclo de vida. Las acciones más comunes como el manejo de solicitudes entrantes, la decodificación de los parámetros, la modificación y el almacenamiento de estado de páginas web como salida hacia el navegador, todas se ejecutan durante el ciclo de vida que tiene la aplicación web. Algunas ventanas de trabajo web ocultan los detalles del ciclo de vida mientras que otros requieren ser manejados manualmente.

Por defecto, JavaServer Faces, las acciones que se presentan en los ciclos de vida son manejadas de forma automática en una página web. Es posible acceder a los diferentes escenarios del ciclo de vida y poder modificar e incluso ejecutar diversas acciones si se requiere.

No se requiere conocer todo el ciclo de vida de la aplicación para quien está comenzando a desarrollar aplicaciones JavaServer Faces.

A continuación se describe como comienza y termina el ciclo de vida de una aplicación JavaServer Faces. Inicialmente el cliente hace una petición a la aplicación a través de una página web, y la respuesta del servidor se da a través de una página web. Como se ve el ciclo consta de ejecutar y renderizar dos fases principales.

Durante la fase de ejecución se realizan varias acciones:

1. La capa de la vista de la aplicación es construida o restaurada.
2. Los valores almacenados en los parámetros de petición son aplicados.
3. Las validaciones y las conversiones son ejecutadas desde los valores de los componentes.
4. Los beans lógicos de respaldo son actualizados con los valores de los componentes.
5. Se invoca la lógica de la aplicación.

La creación de páginas

Para este tema vamos a realizar un ejemplo práctico para el desarrollo de una aplicación haciendo uso de JavaServer Faces. Pueden seguirlo en su IDE de desarrollo para que practiquen la realización del ejercicio.

El ejemplo a desarrollar consta de dos páginas, en la primera página se ingresa el nombre de usuario y la contraseña y la segunda página mostrará un mensaje de saludo de bienvenida al usuario. Utilizaremos en IDE de desarrollo NetBeans.

Pasos:

1. Iniciamos creando un nuevo proyecto: Nuevo proyecto / Java Web / Web Application, seleccionamos el servidor, utilizaremos: Glass Fish; utilizaremos la sexta versión de Java EE (Java EE 6 Web), y por ultimo seleccionamos Java Server Faces y luego clic en Terminar.

2. En el archivo web.xml, en este se modifica para que las páginas sean procesadas por el servlet de JSF, de la siguiente forma:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Ejemplo JavaSeverFaces</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figura 2 Archivo web.xml, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

3. El siguiente paso es crear el beans el cual estará asociado al modelo lógico de usuario y se encargara de guardar la información. Los atributo nombre y contraseña serán las propiedades que también generan los métodos get y set del beans.

```java
package com;

//Bean parte Lógica de la app JSF

public class Usuario {
    private String nombre;
    private String contrasena;

    //Método Nombre
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    //Método contraseña
    public String getContrasena() {
        return contrasena;
    }
    public void setContrasena(String contrasena) {
        this.contrasena = contrasena;
    }
}
```

Figura 3. Archivo Beans, métodos get y set. Código genera por el autor desde IDE Netbeans
Fuente: Propia.

4. Posteriormente creamos la página index.xhtml, que será la página principal de la aplicación y la que se cargara según el mapeo definido en el archivo web.xml

```
<welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>
```

Y el resultado es el que tenemos a continuación.

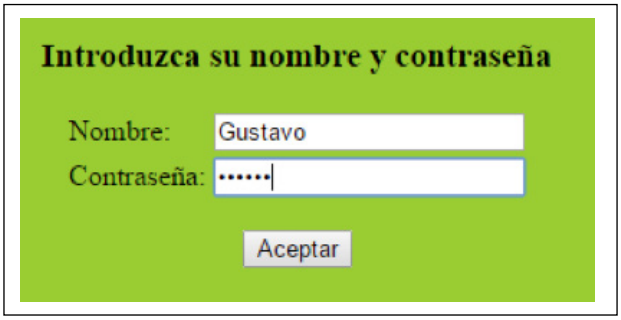


Figura 4. Ingresando datos. Imagen generado por el autor
Fuente: Propia.

El código del index.xhtml es el siguiente:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Introducir datos del Usuario</title>
    </h:head>
    <h:body>
        <center>
                <h:form>
                    <style type="text/css">
                        body {background:yellowgreen}
                    </style>
                    <h3>Introduzca su nombre y contraseña</h3>
                    <table>
                        <tr><td>Nombre:</td>
                            <td><h:inputText value="#{usuario.nombre}"/></td>
                        </tr>
                        <tr><td>Contraseña:</td><td>
                                <h:inputSecret value="#{usuario.contrasena}"/></td>
                        </tr>
                    </table>
                    <p>
                        <h:commandButton value="Aceptar" action="saludo.xhtml"/>
                    </p>
                </h:form>
            </center>

    </h:body>
</html>
```

Figura 5. Archivo index.xhtml. Código generado por el autor desde el IDE de Netbeans
Fuente: Propia.

5. Luego creamos el archivo de la página de saludo saludo.xhtml, el cual tendrá el siguiente código.

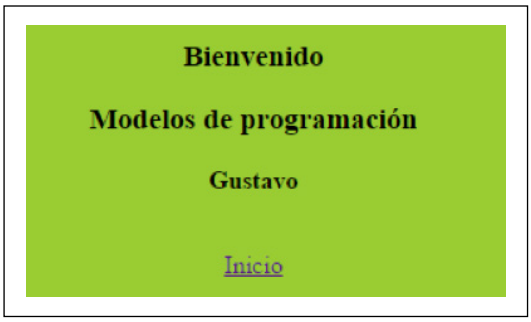


Figura 6. Salida archivo saludo.xhtml.
Fuente: Propia.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Archivo de saludo al usuario</title>
    </h:head>
    <h:body>
        <!-- Establece color del fondo yelloagreen -->
        <style type="text/css">
            body {background:yellowgreen}
        </style>
        <center>
            <h3>Bienvenido</h3>
            <h3>
                Modelos de programación
            </h3>
            <h:form>
                <b><h:outputText value="#{usuario.nombre}"/></b>
            </h:form>
            <br/><br/>
            <a href="index.xhtml">Inicio</a>
        </center>
    </h:body>
</html>
```

Figura 7. Código de salida archivo saludo.xhtml. Código generado por el autor desde el IDE de Netbeans.
Fuente: Propia.

6. Continuamos creando el archivo faces-config.xml, este archivo es el encargado de la configuración de la aplicación JavaServes Faces es donde se realiza el Mapping de FacesServlet. Es necesario registrar en beans dentro del archivo para que sea usado posteriormente. Se diseña e implementa la navegación de la aplicación en este archivo, el cual tendrá el siguiente código.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <navigation-rule>
        <from-view-id>/index.xhtml</from-view-id>
        <navigation-case>
            <from-outcome>exito</from-outcome>
            <to-view-id>/saludo.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>Usuario</managed-bean-name>>
        <managed-bean-class>com.Usuario</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

Figura 8. Mapping de FacesServlet faces-config.xml. Código generado por el autor desde el IDE de Netbeans
Fuente: Propia.

Segundo ejemplo de creación de páginas

Uso de elementos Select y Option en JavaServer Faces (JSF).

En este ejemplo, vamos a crear una aplicación donde implementaremos Select y Option con JavaServer Faces (JSF), con una entrada de texto mediante el uso de inputText, adicionalmente tendrá una lista desplegable que le permitirá elegir una o varias opciones, el estudiante puede seguir el ejemplo e implementarlo en el IDE de práctica Netbeans o Eclipse.

El ejercicio consta de dos páginas que se describen a continuación.

Index.xhtml: en esta página se le pedirá al usuario unos datos de entrada, donde digitará el nombre, apellido y la edad y el país lo seleccionará de una lista que se le suministrará.

Saludo.xhtml: esta página de la bienvenida, mostrando los datos que seleccionamos inicialmente.

Index.xhtml es código es el siguiente:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <center>
        <h:form>
          <style type="text/css">
            body {background:orange}
          </style>
          <h3>Introducir sus Datos</h3>
          <table>
            <tr>
              <td>Nombre:</td>
              <td><h:inputText value="#{Usuario.nombre}"/></td>
            </tr>
            <tr>
              <td>Apellido:</td>
              <td><h:inputText value="#{Usuario.apellido}"/></td>
            </tr>
            <tr>
              <td>Edad:</td>
              <td><h:inputText value="#{Usuario.edad}"/></td>
            </tr>
          </table>
          <p>
            <td>Elija su País</td>
            <SELECT  name="Elegir Componente">
              <OPTION>Argentina</OPTION>
              <OPTION>Brasil</OPTION>
              <OPTION>Chile</OPTION>
              <OPTION selected ="componente5">Colombia</OPTION>
              <OPTION>Ecuador</OPTION>
              <OPTION>Mexico</OPTION>
              <OPTION>Paraguay</OPTION>
              <OPTION>Peru</OPTION>
              <OPTION>Venezuela</OPTION>
            </SELECT>
          </p>
          <p>
            <h:commandButton value="Aceptar" action="saludo.xhtml"/>
            <h:commandButton value="Reiniciar" type="reset"/>
          </p>
        </h:form>
      </center>
  </h:body>
</html>
```

Vista del formulario que solicita los datos de entrada

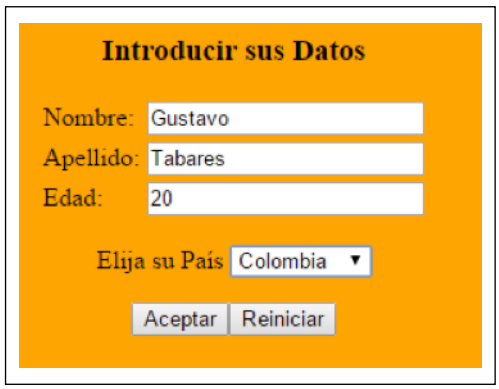


Figura 9. Formulario ingreso de datos
Fuente: Propia.

El archivo **saludo.xhtml** se encarga de mostrarla salida con los datos capturados el código es el siguiente.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <!--color de fondo-->
        <style type="text/css">
            body {background:orange}
        </style>
        <center><h3>Bienvenido</h3>
            <h:form>
                <h:outputText value="#{usuario.nombre} #{usuario.apellido}"/>
                <h:outputText value="#{usuario.sexo}"/>
            </h:form>
            <br/>
            <a href="index.xhtml">Inicio</a>
        </center>
    </h:body>
</html>
```

Figura 10. Código salida archivo saludo.xhtml. Código generado por el autor desde el IDE de Netbeans
Fuente: Propia.

La salida del archivo saludo.xhtml es la siguiente:

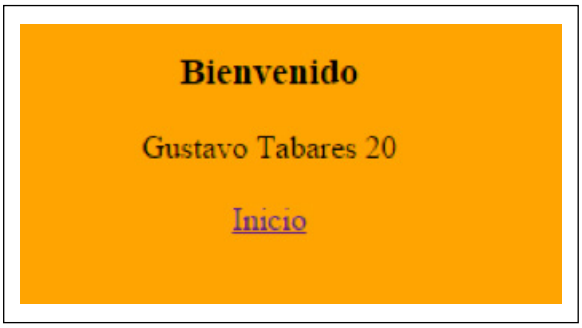


Figura 11. Interfaz de salida archivo saludo.xhtml
Fuente: Propia.

El archivo web.xml, se debe modificar para definir la estructura de páginas a ejecutar

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figura 12. Estructura de navegación web.xml. Código generado por el autor desde el IDE de Netbeans. Fuente: Propia.

El siguiente paso es crear el bean es cual estará asociado con al usuario y nos permitirá almacenar la información, en este archivo se crearan los métodos set y get correspondientes a los atributos del usuario. El archivo tendrá la siguiente estructura.

```java
package com;

//Bean parte Lógica de la app JSF
public class Usuario {
    private String nombre;
    private String apellido;
    private int edad;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

Figura 13 Beans clase usuario. Código generado por el autor desde el IDE de Netbeans. Fuente: Propia.

Ahora en el archivo faces-config.xml, vamos implementar las reglas de navegación, debemos de registrar el bean para que sea usado en el archivo faces-config.xml, el código del archivo es el siguiente.

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
 <navigation-rule>
        <from-view-id>/index.xhtml</from-view-id>
        <navigation-case>
            <from-outcome>exito</from-outcome>
            <to-view-id>/saludo.xhtml</to-view-id>
        </navigation-case>
   </navigation-rule>

    <managed-bean>
        <managed-bean-name>usuario</managed-bean-name>>
        <managed-bean-class>com.Usuario</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

Figura 14. Mapping de FacesServlet faces-config.xml. Código generado por el autor desde el IDE de Netbeans. Fuente: Propia.

2

Unidad 2

Servicio web

Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Hoy el día encontramos mucha tendencia en el diseño del software hacia una programación más modular, donde la aplicaciones esta compuestas de una serie de componentes reutilizables que podemos entenderlos como servicios, estos pueden estar distribuidos a los largo de una red de computadoras generando información y recursos que a la vez pueden ser consumidos por alguna aplicaciones que los requieran.

Los servicios web nos permitirán distribuir nuestra aplicación a través de la Internet, donde una aplicación puede hacer uso de los servicios ofrecidos por cualquier servidor de internet que comparta un servicio.

U2

Metodología

El estudiante encuentra en el inicio toda la información sobre el módulo, que le permite abordarlo y comprender su dinámica de funcionamiento.

Para cada unidad el estudiante cuenta con un texto que se conoce como guía de actividades, el cual semana a semana le informa qué actividades tiene para la semana y qué debe hacer. Así mismo le informa las actividades calificables en su módulo. En dicho texto se incluyen los objetivos de aprendizaje de la unidad.

La cartilla le permitirá abordar el tema de los web services bajo la programación Java. Se recomienda leer cuidadosamente y en forma secuencial la semana para tener una compresión adecuada del tema.

U2 Desarrollo temático

Servicio web

Un servicio web es un componente al que podemos acceder a través de protocolos web estándar, haciendo uso del XML, SOAP, WSDL y UDDI sobre protocolos de la Internet como (HTTP, SMTP, TCP, FTP) como intermediario para el intercambio de información.

XML: permite describir los datos que ofrece el servicio.

SOAP: es el encargado de la transferencia de los datos.

WSDL: es el encargado de describir los servicios disponibles en el web service.

UDDI: es el encargado de conocer cuáles son los servicios disponibles para ser consumidos.

Un servicio web es una colección de procedimientos que son compartidos a través de páginas de internet, son llamados desde cualquier lugar de internet, esta invocación es totalmente independiente de la plataforma y del lenguaje de programa en el que se haya programado el servicio.

Cuando nos conectamos a la Internet y hacemos una petición al servidor web desde el navegador, el servidor devuelve la página solicitada, con la información para ser visualizada por el usuario, y difícil de entender por una máquina, los servicios web se ofrecen con un formato estándar para que sea entendido fácilmente por otra aplicación.

Los servicios web son componentes de aplicaciones totalmente distribuidas que están disponibles a las aplicaciones de forma externa. Pueden ser utilizados para integrar aplicaciones en diferentes lenguajes y plataformas, debido a los estándares web de algunos servicios web.

Características de los servicios Web

- Cualquier servicio web debe ser posible su acceso a través de la web. Para esto debe estar basado en protocolos de transporte estándar como el HTTP, y los mensajes se deben codificar en un lenguaje que pueda ser interpretado fácilmente por el cliente.
- El servicio debe tener una descripción de sí mismo, esto para que las aplicaciones que lo invoquen sepan la función determinada de un servicio web.

- Debe ser de fácil localización, se debe desarrollar un mecanismo de fácil localización del servicio web que realiza una determinada función. De esta forma podemos localizar a través de una aplicación el servicio de forma automática, sin saber con antelación como es el servicio.

Un servicio web es un componente de software que se ofrece a través de una página por medio de un endpoint y es accesible a través de la red. Los servicios web tiene la funcionalidad de ser producidos y consumidos intercambiando información a través de mansajes invocados por peticiones y respuestas en forma de documentos auto-contenidos.

A nivel técnico los servicios web pueden implementarse de varias formas para ser consumidos por los usuarios, tenemos dos tipos de servicios web: los servicios Web "grandes" (big Web Services), que también se llaman SOAP, y los servicios Web RESTful.

Servicios web Soap

Los servicios web SOAP, utilizan mensajes bajo el estándar XML y comunicarse entre páginas con el servicio SOAP (Simple Object Access Protocol), a través de un XML que define la estructura y formato de los mensajes que se comunican, las operaciones que se brindan por el servicio, estas están escritas en WSDL (Web services Description Language), este lenguaje está basado en el lenguaje XML que permite definir las interfaces sintácticamente.

Hoy en día los IDE de desarrollo como Netbeans y Eclipse, etc. Permiten reducir la complejidad para el desarrollo de aplicaciones que implementen servicios web para los usuarios, estos están bajo un formato SOAP (formato de mensajes) y WSDL (lenguaje de definición de interfaces) Se que se ha extendido y utilizado por muchos IDE de desarrollo.

El diseño para la implementación de un servicio basado en SOAP este permite implementar una especie de contrato formal en el cual se describe la interfaz que ofrece el servicio Web de la aplicación. WSDL permite utilizarse para dar una descripción los detalles del contrato, donde se puede incluir mensajes, las operaciones a realizar, los bindings, y la localización del servicio Web para su consumo. Los requerimientos no funcionales deben de tenerse en cuenta, como son las transacciones de la aplicación, la necesidad de mantener el estado (addressing), la seguridad y coordinación de la ejecución del servicio.

Servicios web (RESTful)

Otro tipo de servicio web es el RESTful (Representational State Transfer Web Services), son ideales para espacios web de integración básicos ad-hoc. Este servicio se integra mejor con HTTP que los servicios basados en SOAP, debido a que no es necesario mensajes XML o las definiciones del servicio en forma de fichero WSDL.

El servicio web RESTful está basado en estándares muy utilizados como son HTTP, MIME, SML, URI, posee una infraestructura muy práctica (ligera), lo que permite que los servicios se vallan construyendo con herramientas de forma mínima. Debido a esto es tipo de servicios es muy económico y no tiene muchos obstáculos para su implementación.

Arquitectura de los servicios web

Los servicios Web entre aplicaciones están basados en una arquitectura orientada a servicios que permite crear una definición abstracta de un servicio, para poder proporcionar la información e implementar de forma concreta dicho servicio, que permita publicar y localizar un servicio a través aplicaciones, seleccionar una instancia de un servicio, y utilizar el servicio con una seguridad y calidad. Es posible desacoplar la implementación del servicio Web y su uso por parte de un cliente. Las implementaciones concretas del servicio pueden desacoplarse a nivel de lógica y transporte. La siguiente figura muestra en detalle la arquitectura orientada a servicios web.

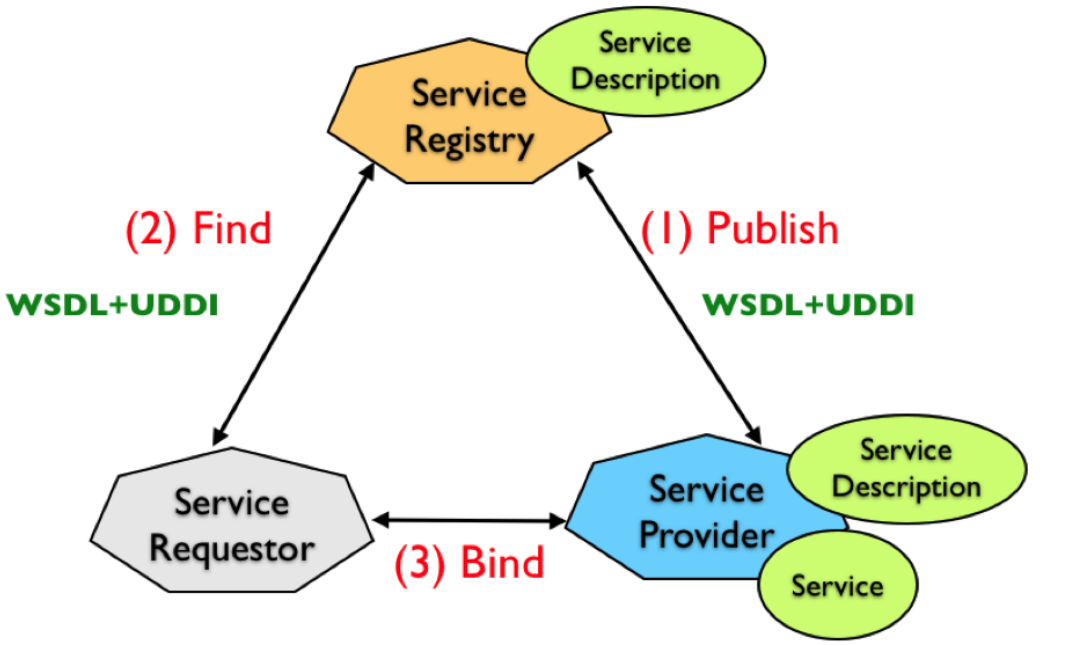


Figura 1. Arquitectura de los Servicios Web
Fuente: http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html

La página que ofrece el servicio define una descripción abstracta del servicio utilizando el lenguaje WSDL (Web Services Description Language) Lenguaje de descripción de servicios web. Lo que permite la creación de un servicio concreto a partir de la dicha descripción. Esta información descriptiva del servicio puede ser publicada en un servicio de registro como UDDI (Universal Description, Descovery and Integration). Un consumidor de un servicio puede utilizar el servicio de registro para localizar una descripción de un servicio, lo que le permitirá seleccionar y hacer uso de los métodos, eventos y funciones implementadas del servicio.

El documento WSDL describe de forma abstracta el servicio como un PortType, es este una instancia concreta de un servicio se define mediante un elemento port de un WSDL, el cual consiste en una serie de combinaciones de un PortType, un binding de codificación y transporte, más la dirección.

Servicios Web Soap y Java

Características de los componentes de un servicio web.

- Los métodos se implementan bajo una interfaz que se describe con un lenguaje de definición de interfaces (WSDL). Estos se implementan con EJB o un componente web JAX-WS.
- Un servicio web puede estar publicado en varios registros para su posterior despliegue.
- Cualquier servidor que este con Java EE, permite el despliegue de un servicio web que tenga una funcionalidad descrita WSDL.
- Los servicios se consumen en tiempo de ejecución (run-time).

Java cuenta con un API, la API Java para XML basada para los Web Services (JAX-WS), es una parte importante de la plataforma JavaEE, tiene como función la simplificación de las tareas de desarrollo de los web services utilizando tecnología Java. La versión cuenta con soporte para protocolos SOAP 1.1, SOAP 1.2, XML, permitiendo utilizar también otros protocolos adicionales juntos con HTTP.

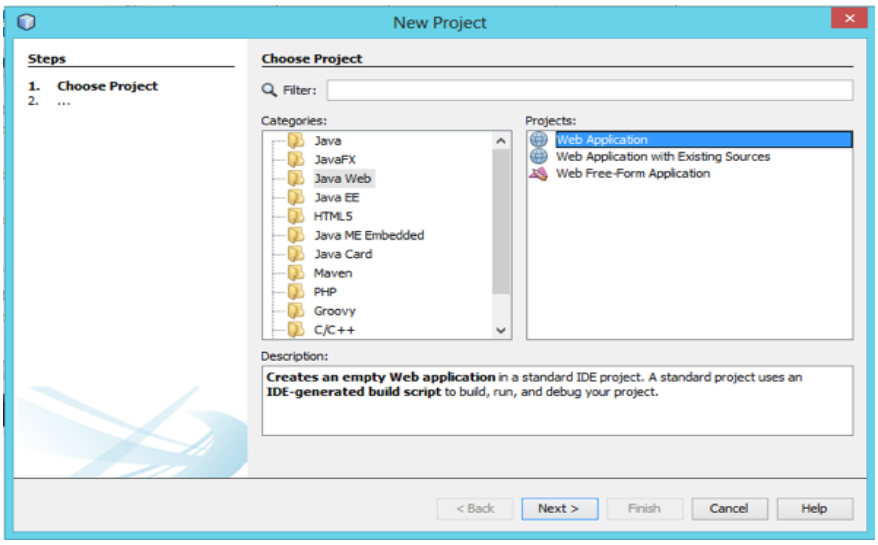
JAX-WS utiliza JAXB 2.0 para enlazar datos y soportar la personalización de los servicios que se vayan creando.

Mediante un ejemplo crearemos un web service y este será consumido por tres aplicaciones clientes de la siguiente forma:

- Una clase Java dentro de una aplicación Java SE.
- Un Servlet dentro de una aplicación Web.
- Una página JSP dentro de una aplicación Web.

Crear un Web Service

Para crear un Web service lo primero es elegir el contenedor a utilizar "Deployment Container" para desplegar un web servicies se puede hacer en un contenedor Web o en un contenedor EJB (Enterprise JavaBeans).



Seleccionamos File – New Project, seleccionamos en categorías "Java web" y en Projects seleccionamos Web Apllication.

Figura 2. Creación aplicación Web imagen generada por el autor desde el IDE Netbeans Fuente: Propia.

Clic en siguiente, y asignamos el nombre al proyecto "AplicacionCalculadoraWS"

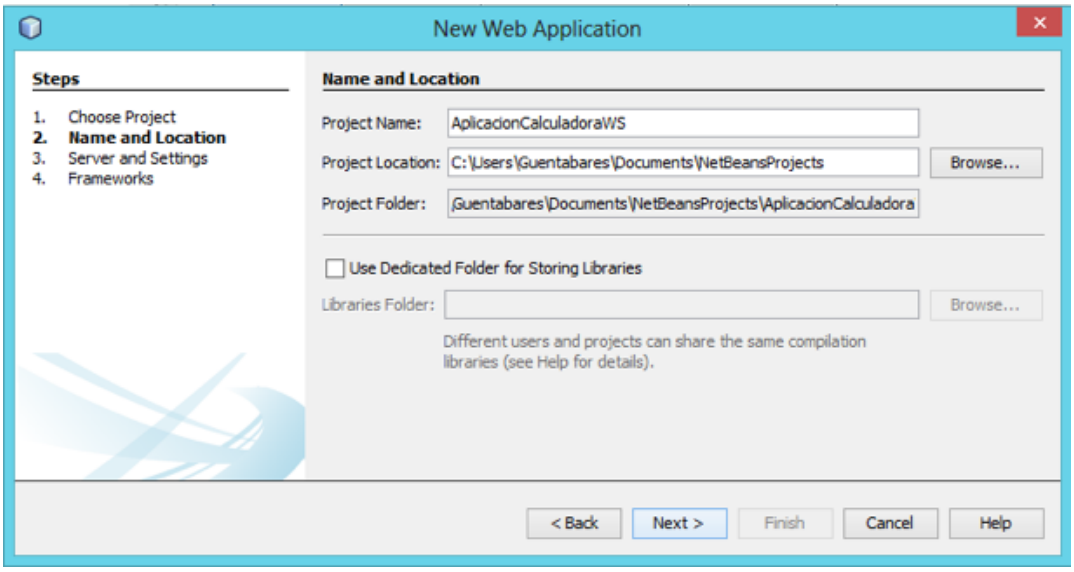


Figura 3. Definiendo nombre aplicación Web imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos el servidor GlassFish, y Java EE Version seleccionamos la Java EE7.

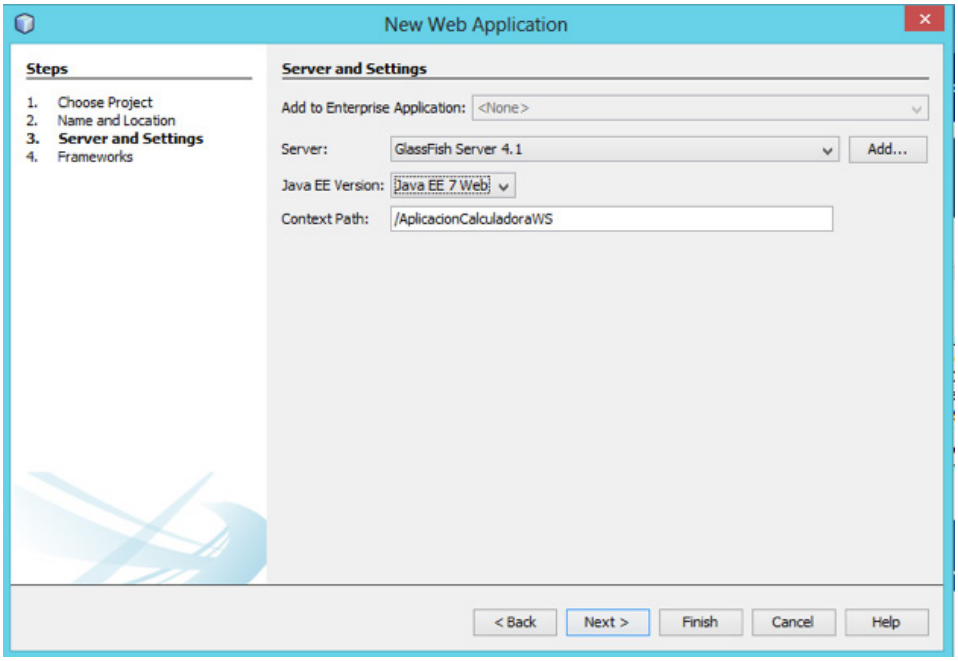


Figura 4. Selección servidor GlassFish imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Clic en finish, con esto ya tenemos el contenedor para el web service.

El paso siguiente es crear el web service, para esto damos clic con el botón derecho del mouse sobre el proyecto y seleccionamos new – Web Service…

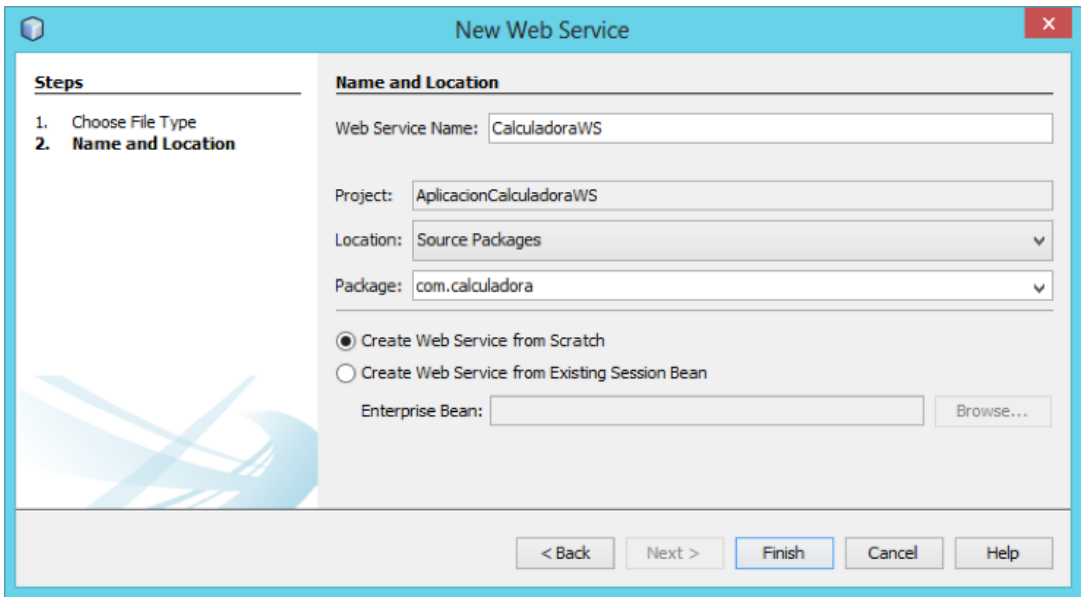


Figura 5. Crear Web Services imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Asignamos el nombre Web Service Name "CalculadoraWS" y creamos el paquete com.calculadora y clic en finish.

Si observamos la estructura de archivos del proyecto vemos que nos ha creado una carpeta Web Services.

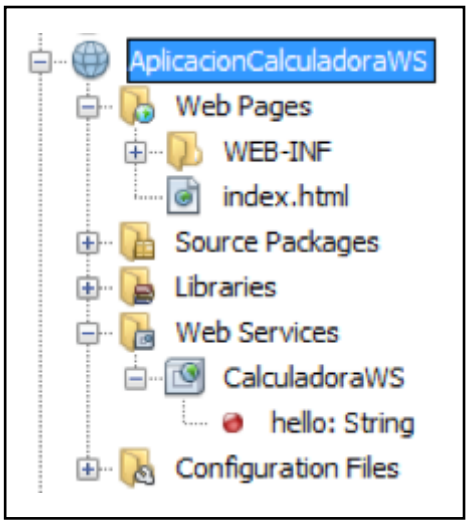


Figura 6. Estructura de archivos AplicacionCalculadoraES imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Y nos ha generado el código del web services, como vemos en la gráfica.

```java
package com.calculadora;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Guentabares
 */
@WebService(serviceName = "CalculadoraWS")
public class CalculadoraWS {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

Figura 7. Código del Web Services código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Vamos a añadir una operación al web service, cambiamos a modo Desing ubicado en la parte superior del código. Con esto no muestra el modo de diseño del web service.

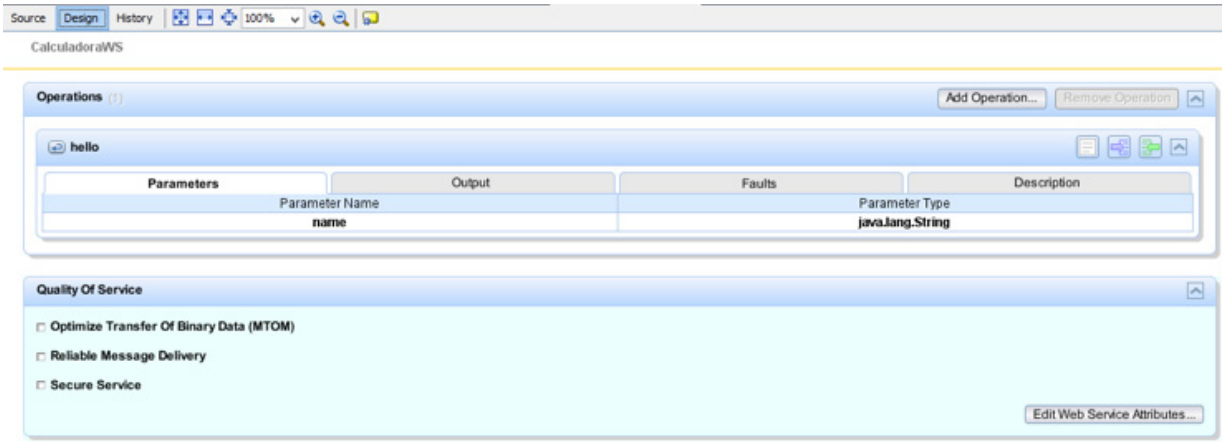


Figura 8. Diseño del Web Services imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Para crear una nueva operación damos clic en el botón Add Operation… lo que nos arroja la siguiente ventana, donde damos en nombre "add" y en Return Type: escribimos int que es el valor que va a arrojar el web service.

En la pestaña parameters se adicionan los valores, en este caso vamos a realizar una operación de suma.

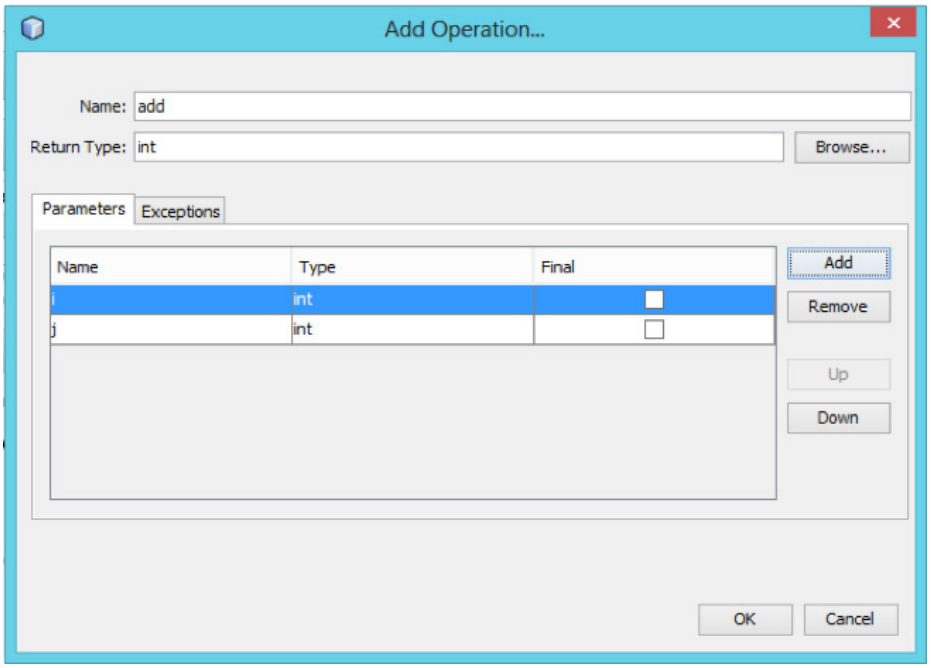


Figura 9. Definición del método y atributos del Web Services imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto ya tenemos una operación en nuestro web service, como se ve en la gráfica.

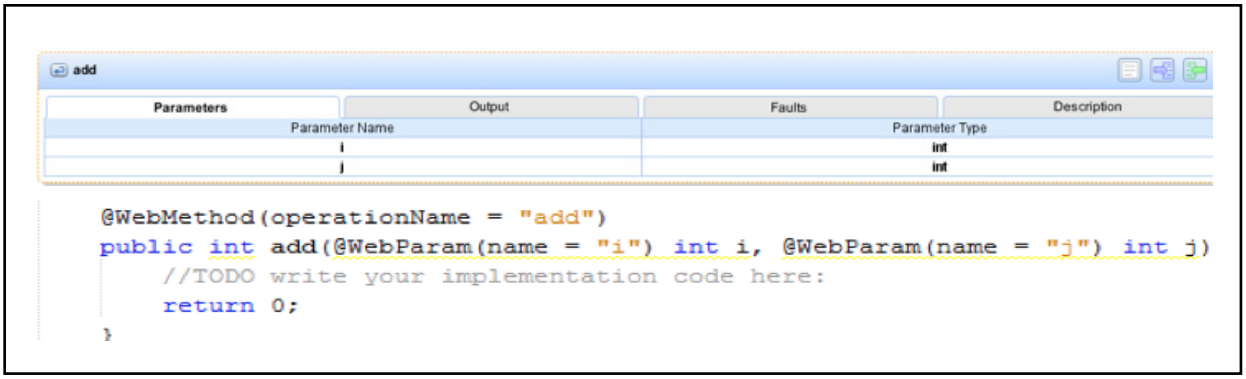


Figura 10. Atributos enteros del método del Web Services código creado por el autor desde el IDE Netbeans
Fuente: Propia.

También ha creado el código de la operación en el source, donde debemos modificarlo para adicionar el método para sumar las variables.

```
Method(operationName = "add")
ic int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j
int k = i + j;
return k;
```

Figura 11. Creación de la programación del método add código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto hemos creado un Web Services que recibe dos números y devuelve el resultado.

Para realizar un test web service, y probar si funciona nuestro ws, damos clic derecho sobre el proyecto y seleccionamos la opción Deploy. Después damos clic con el botón derecho del mouse sobre el web service y seleccionamos test web sevice.

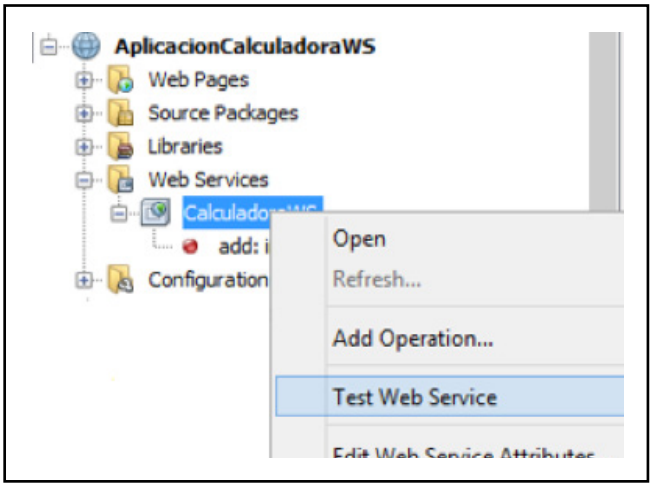


Figura 12. Test del Web Service imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Para tener como salida la vista del web service que se ha creado.

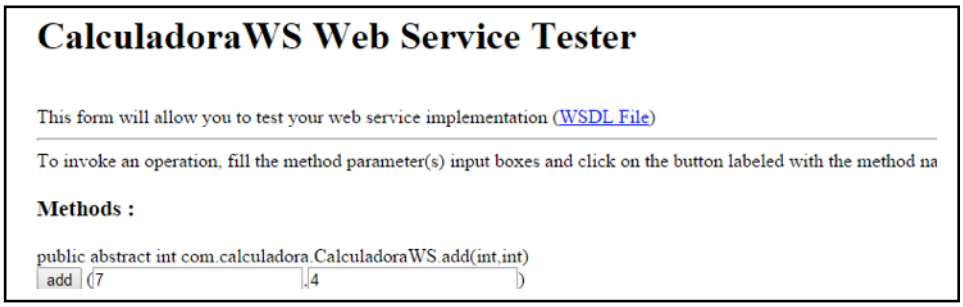


Figura 13. Resultado del Web Service imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Damos clic en el botón add y tendremos la descripción del web service, con la respuesta del SOAP Request y SOAP Response.

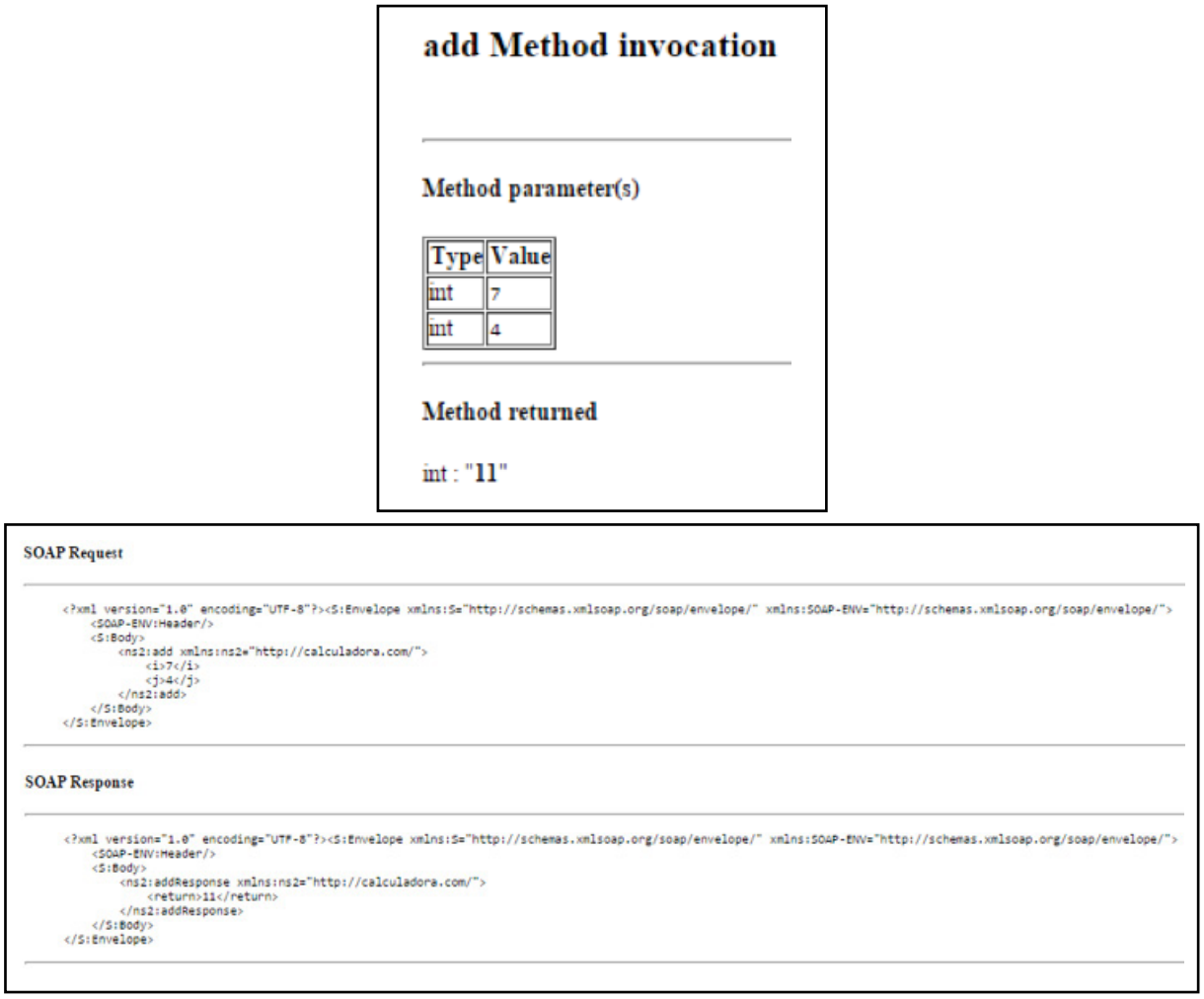


Figura 14. Salida métodos Request y Response, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto ya tenemos creado el web services, a continuación crearemos un cliente para consumir los métodos y funciones que tiene el web service claculdoraWS.

Web Services cliente.

Vamos a crear tres clientes distintos que consuman el web service:

- Una clase Java en una Aplicación Java SE.
- Un servlet en una Aplicación Web.
- Una Página JSP en una Aplicación web.

Primer cliente: Una clase Java en una Aplicación Java SE.

Los pasos para crear se describen a continuación:

1. Crear aplicación Java.
2. Crear Clase Java.
3. Crear un Cliente Web Service.
4. Consumir el Web Service.

Seleccionamos file – New Project, seleccionamos de la categorías Java y de Projects la opción Java Application.

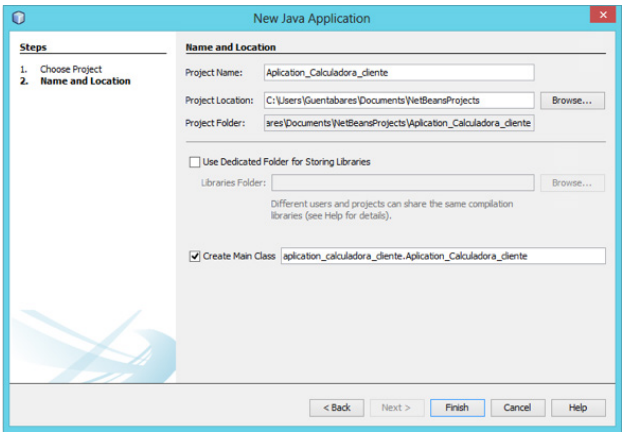


Figura 15. Creación de la aplicación cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Asignamos el nombre de Aplication_Calculadora_cliente. Lo que nos dará como respuesta la clase Java.

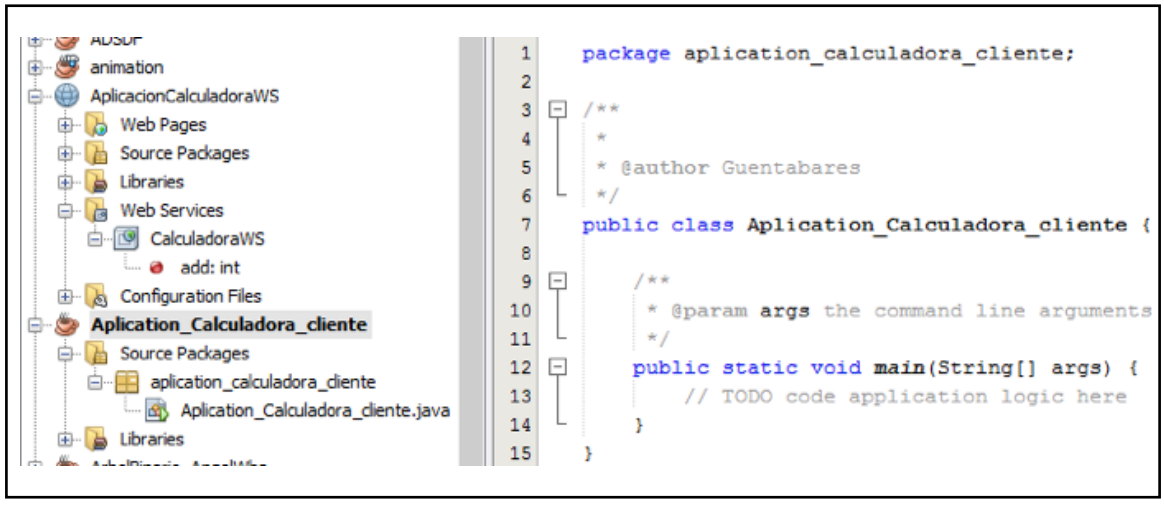


Figura 16. Aplicación calculadora cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto damos clic con el botón derecho del mouse sobre el proyecto "Aplication_Calculadora_cliente" y seleccionamos new – Web Service Client…

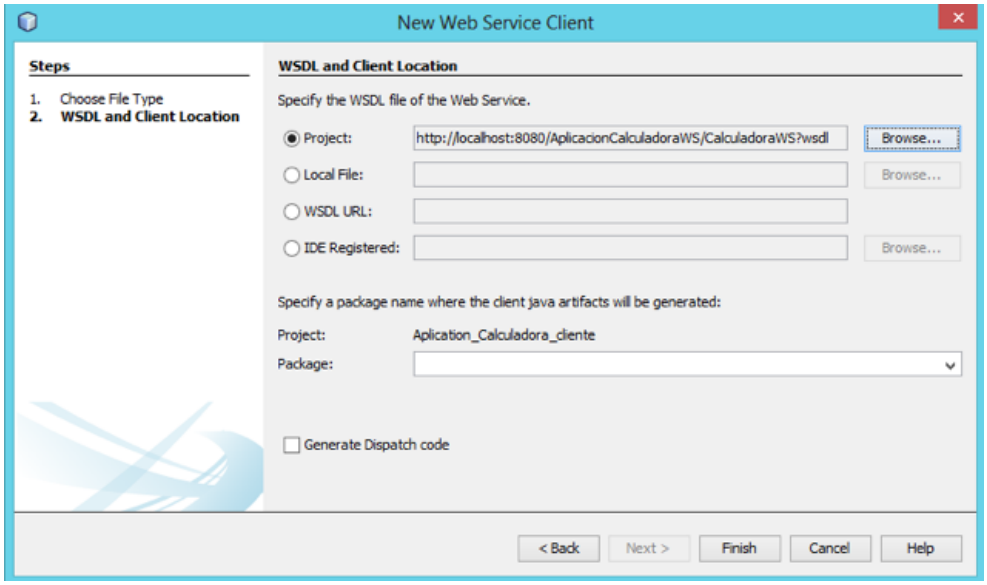


Figura 17. Definición del WSDL Cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos Project y en el botón browse buscamos el web service CalculadoraWS. Y clic en Finish.

Observemos que el proyecto del web service client integra el WSDL (lenguaje de definición de interfaces).

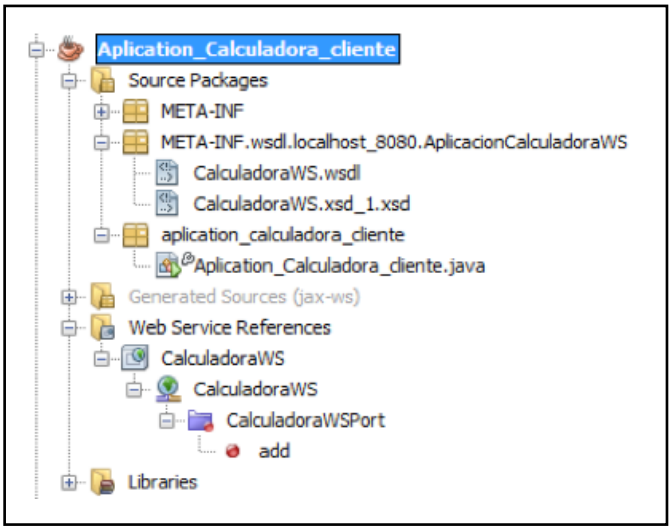


Figura 18. Estructura aplicación calculadora cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Observemos como el proyecto Aplication_Calculadoraa_cliente, adiciona los paquetes con WSDL Lenguaje de descripción de servicios web. Lo que permite la creación de un servicio concreto a partir de la dicha descripción. También adiciona el Web service References, con el web service "CalculadoraWS" y sus métodos creados en nuestro proyecto web anteriormente para ser consumidos.

Para consumir el método del Web Service solo arrastramos el método dentro del main y asignamos valores a las variables para probar la funcionalidad del web service. El código de la clase quedaría de la siguiente forma, con el web service incorporado para ser consumido dentro de la clase Java:

```
public class Aplication_Calculadora_cliente {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        add(7, 4);

    }
    private static int add(int i, int j) {
        com.calculadora.CalculadoraWS_Service service = new com.calculadora.CalculadoraWS_Service();
        com.calculadora.CalculadoraWS port = service.getCalculadoraWSPort();
        int resultado = port.add(i, j);
        System.out.println("El Resultado es "+resultado);
        return 0;
    }


}
```

Figura 19. Consumiendo el método add, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ejecutamos la clase Aplication_Calculadora_cliente para probar la salida del web service.

Segundo cliente: un Servlet en una aplicación.

Los pasos para crear se describen a continuación:

1. Crear aplicación Web.
2. Crear un Servlet.
3. Usamos el Servlet para consumir el Web Service.

Seleccionamos File – New Project, seleccionamos en categorías "Java web" y en Projects seleccionamos Web Apllication.

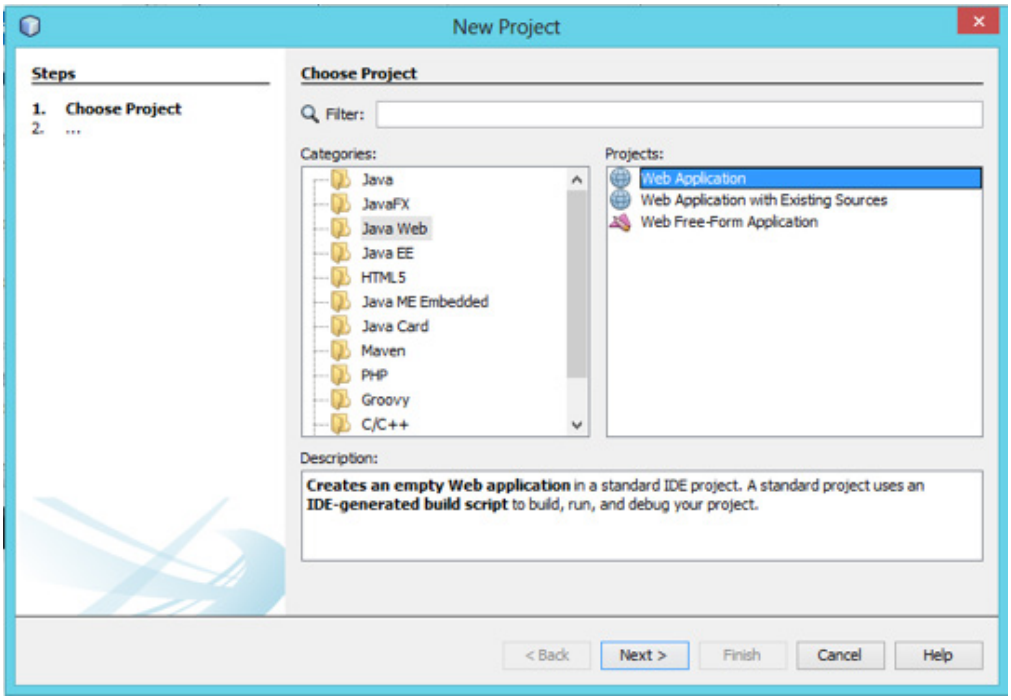


Figura 20. Creación de la una aplicación Web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Le asignamos el nombre al proyecto de ProyectoCalculadoraWSServeltCliente y clic en siguiente.

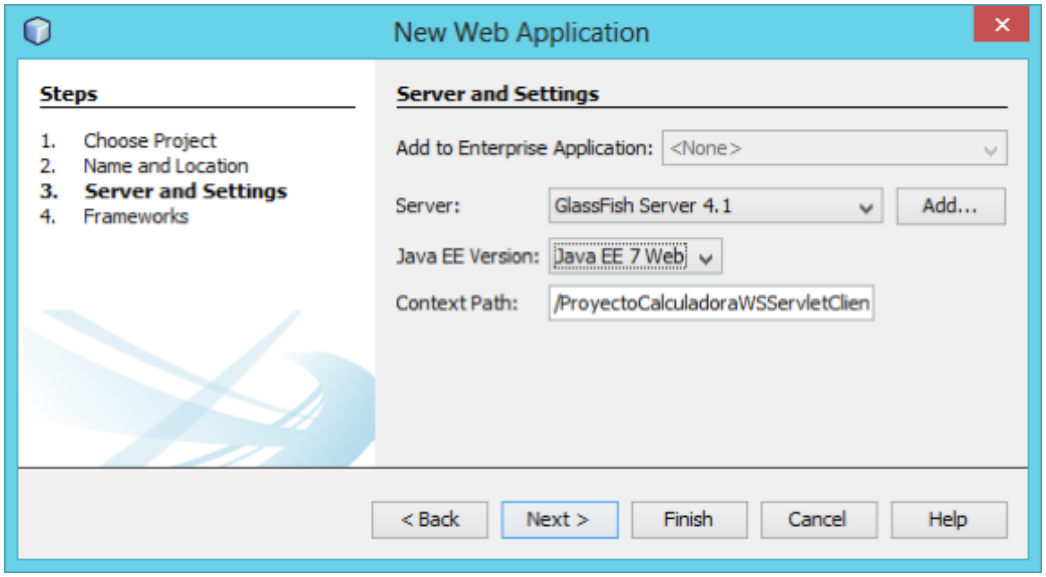


Figura 21. Definición del servidor web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Dejamos por defecto aparece en la ventana, clic en finalizar, y se crea el proyecto

Ahora para consumir el web services, damos clic con el botón derecho del mouse, seleccionamos new – Web Service Client.

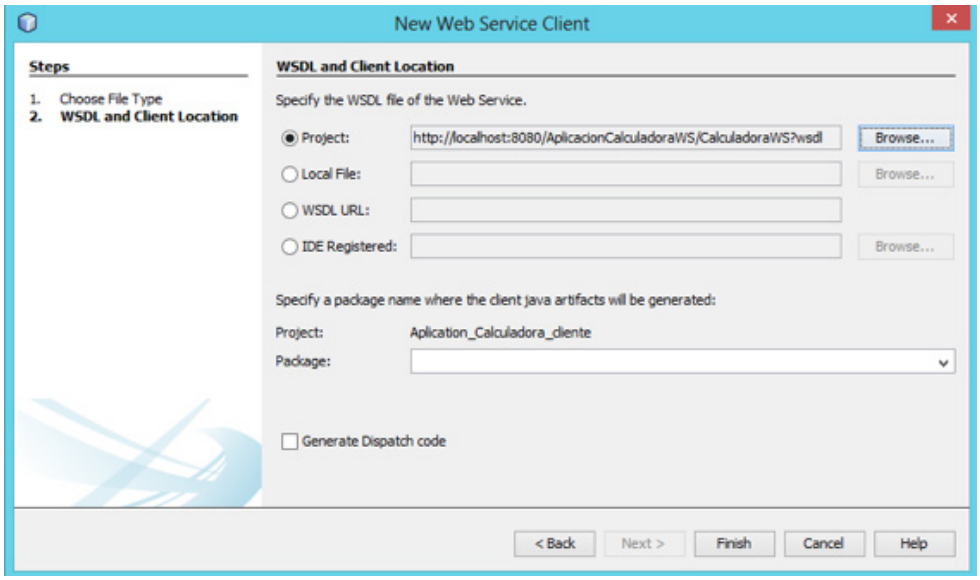
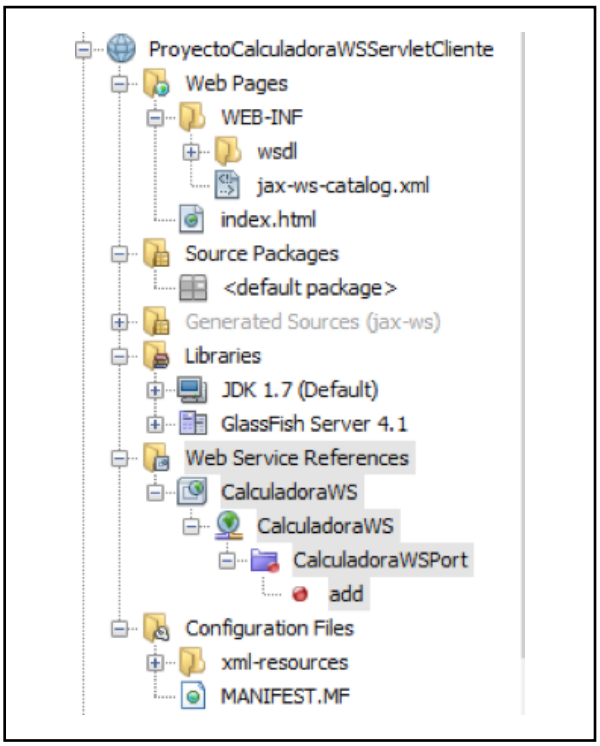


Figura 22. Consumiendo el Web Services, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.



Seleccionamos Project y en el botón browse buscamos el web service CalculadoraWS. Y clic en Finish.

Observemos que el proyecto del web service client integra el WSDL (lenguaje de definición de interfaces).

Figura 23. Estructura de la aplicación Web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ya tenemos integrado en la aplicación web el web service para ser consumido, como se observa en la estructura del proyecto en la carpeta Web Service References.

Ahora vamos a crear el servlet para poder usar nuestro web service.

Damos clic con el botón derecho sobre el ProyectoCalculadoraWSServletCliente, seleccionamos new – Servlet.

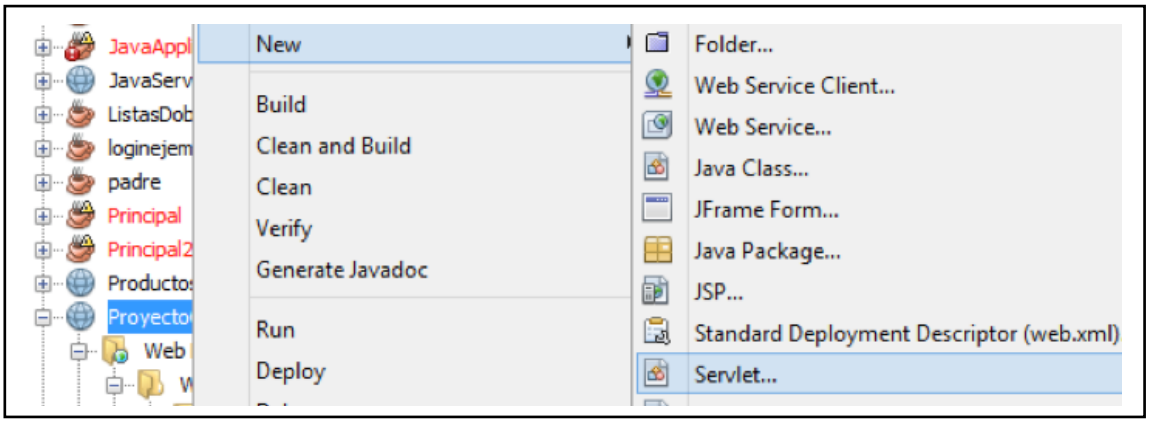


Figura 24. Creación del servlet de la aplicación web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Donde tendremos que darle el nombre "ClienteServlet", creamos el paquete com.calculadora.cliente, dejamos el resto de opciones por defecto y clic en Finish.

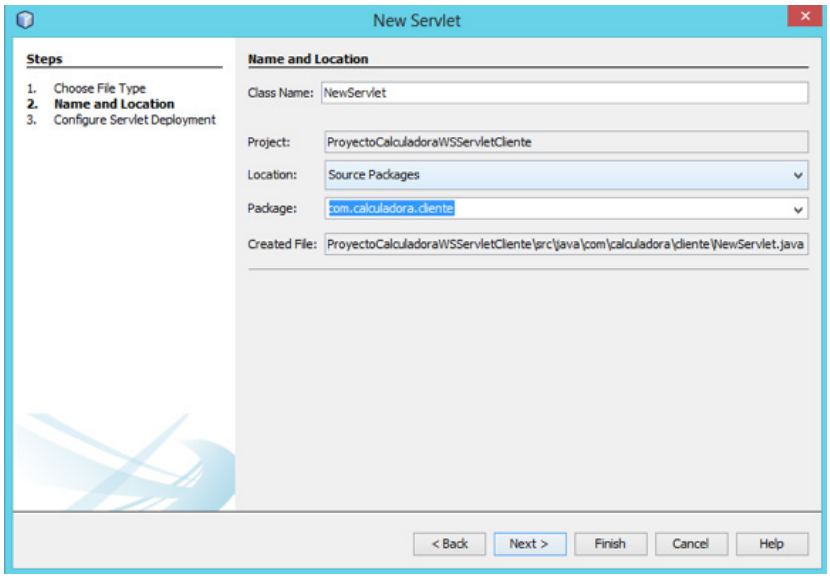


Figura 25. Definiendo la clase Servlet, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto tenemos ya nuestra clase servlet, donde podremos llamar al web service, antes debemos dejar nuestro servlet como punto de entrada a nuestra aplicación, para esto debemos dar clic derecho sobre el proyecto y seleccionar propiedades, en la opción run agregamos el nombre del servlet "ClienteServlet"que creamos y clic en ok.

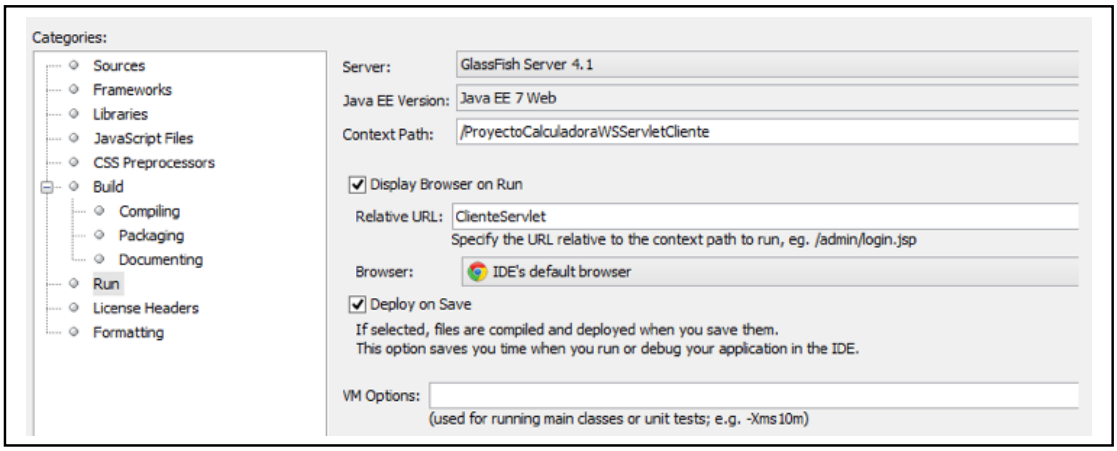


Figura 26. Definiendo el Servlet como punto de entrada de la aplicación, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ahora vamos a agregar el método "add" de nuestro web service, en el servlet que hemos creado dentro del método "processRequest", donde podemos observar código HTML, arrestramos el método "add" dentro de las etiquetas <body> </body> del serlvet, y modificamos el método para que muestra la información de la variable resultado.

El cual debe quedar de la siguiente forma:

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ClienteServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet ClienteServlet at " + request.getContextPath() + "</h1>");
            add(7,4);
            out.println("<h1>Resultado " + resultado + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
    HttpServlet methods. Click on the + sign on the left to edit the code.
    private int add(int i, int j) {
        // Note that the injected javax.xml.ws.Service reference as well as port objects are not thread safe.
        // If the calling of port operations may lead to race condition some synchronization is required.
        com.calculadora.CalculadoraWS port = service.getCalculadoraWSPort();
        resultado = port.add(i, j);
        return resultado;

    }
}
```

Figura 27. Código del Servlet, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto ya tenemos el web service para ser consumido dentro de la aplicación web que hemos creado, dando como resultado lo siguiente.

Servlet ClienteServlet at /ProyectoCalculadoraWSServletCliente

Resultado 11

Tercer cliente: Una aplicación JSP.

Seleccionamos File – New Project, seleccionamos en categorías "Java web" y en Projects seleccionamos Web Apllication.

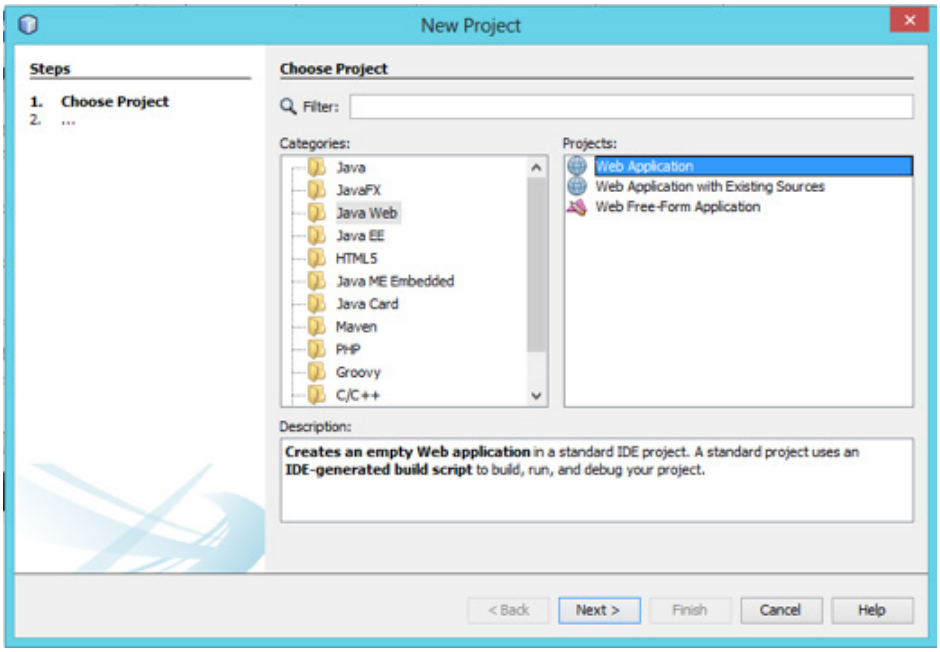
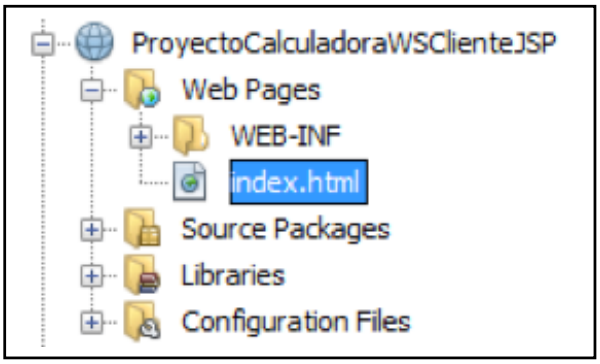


Figura 28. Creación aplicación Web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.



Le asignamos el nombre al proyecto "ProyectoCalculadoraWSClienteJSP", dejamos por defecto el resto de opciones y clic en Finish.

Figura 29. Estructura archivos de la aplicación web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Creamos el web service cliente para consumir los métodos, fusiones y variables en la aplicación JSP.

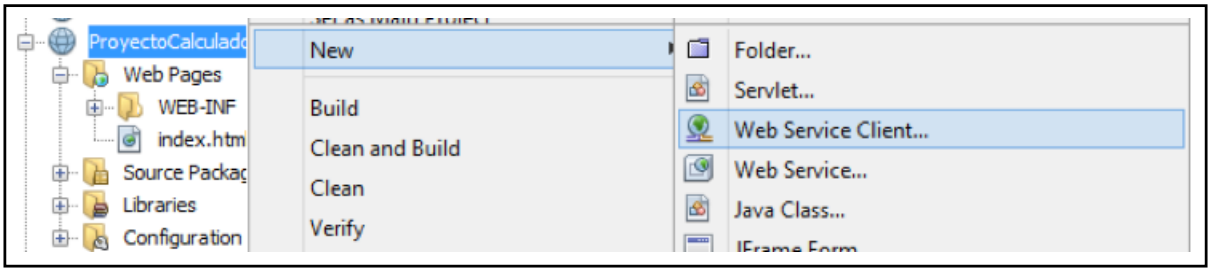


Figura 30. Creación del Web Services Cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ahora para consumir el web services, damos clic con el botón derecho del mouse, seleccionamos new – Web Service Client.

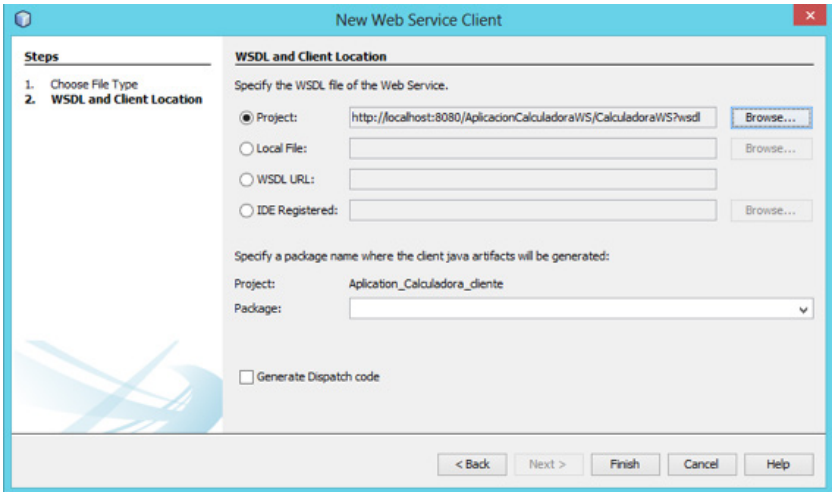


Figura 31. Consumiendo el Web Serves desde el cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos Project y en el botón browse buscamos el web service CalculadoraWS. Y clic en Finish.

Observemos que el proyecto del web service client integra el WSDL (lenguaje de definición de interfaces).

Ya tenemos el webservice incluido en la aplicación JSP.

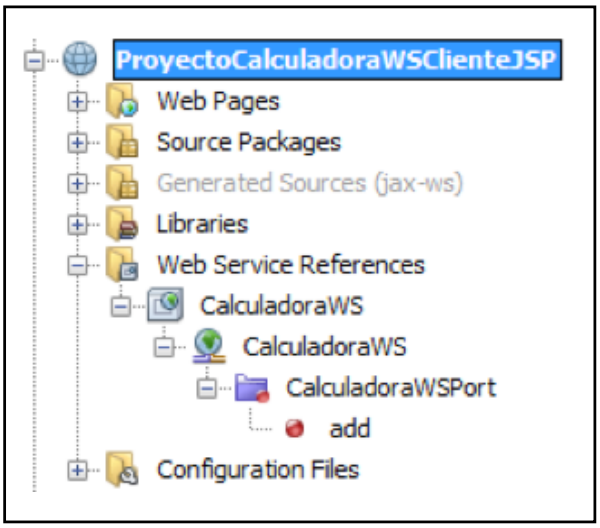


Figura 32. Estructura de archivos de la aplicación, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Para incluir el método del web service en la aplicación JSP, seleccionamos em método "add" y arrastramos a nuestra aplicación.

```
<body>
    <h1>Aplicación JSP</h1
        <%-- start web service invocation --%><hr/>
<%
try {
    com.calculadora.CalculadoraWS_Service service = new com.calculadora.CalculadoraWS_Service
    com.calculadora.CalculadoraWS port = service.getCalculadoraWSPort();
     // TODO initialize WS operation arguments here
    int i = 7;
    int j = 4;
    // TODO process result here
    int result = port.add(i, j);
    out.println("Resultado = "+result);
} catch (Exception ex) {
    out.println("Error al cargar el Web Service");
}
%>
<%-- end web service invocation --%><hr/>
</body>
```

Figura 33. Código del add del Web Services, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Ejecutamos la aplicación "ProyectoCalculadoraWSClienteJSP" y la salida debe ser la siguiente.

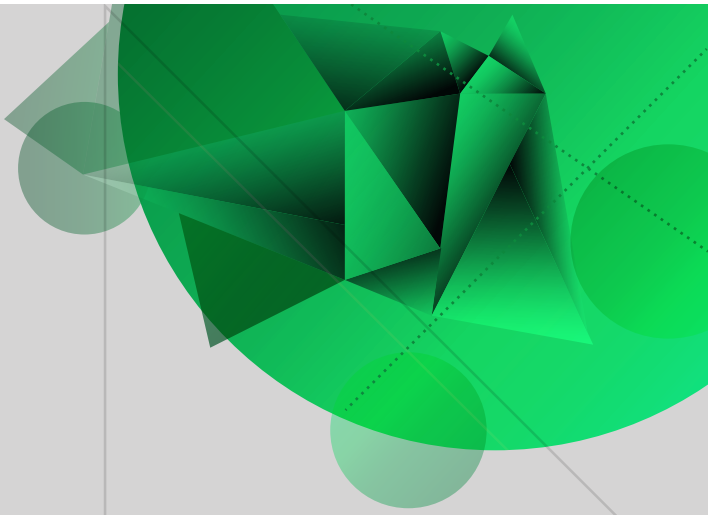
Figura 34. Salida del Web Services, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

El resultado del método consumido del web service.

3

Unidad 3

Beans Enterprise



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Los Beans Enterprise son una de las interfaces de programación de aplicaciones (API) que forman parte del estándar de construcción de aplicaciones empresariales J2EE, teniendo como objetivo encapsular los objetos del lado del servidor que almacena los datos.

U3 Metodología

El estudiante encuentra en el inicio toda la información sobre el módulo, que le permite abordarlo y comprender su dinámica de funcionamiento.

Se recomienda llenar de forma ordenada la lectura de la cartilla para tener una mayor comprensión del tema, no se aconseja saltarse el contenido debido a que la programación orientada a objetos es importante comprender bien el concepto.

Para esta semana hay un material de apoyo video capsula, video diapositiva, que le permitirá afianzar los conceptos y comprender la herencia que se da en la programación orientada a objetos.

Beans Enterprise

Los Session Beans, es importante para saber cómo una aplicación java negocia con el servidor, una de las funciones principales de java Enterprise, es su construcción de componentes de software modulares, formada por APIS.

Imagen 1. Componentes Modulares
Fuente: https://morelos.quadratin.com.mx/Efermerides-el-Dia-del-Rompecabezas/

Java Enterprise Edition (JEE) es la encargada de coordinarlas, esta APIS puede ser comunes o compartidas con Java Estándar Edition (JSE) u otras plataformas de programación que debe cumplir con ciertos requisitos de conformidad para que se dé una integración con JSE.

Algunas APIS están especificadas como componentes únicos para JEE, esta es la que vamos a tratar en esta cartilla Java Enterprise Beans (EJB).

EJBs es la encargada de detallar como los servidores de una aplicación mueven los objetos específicamente JBeans.

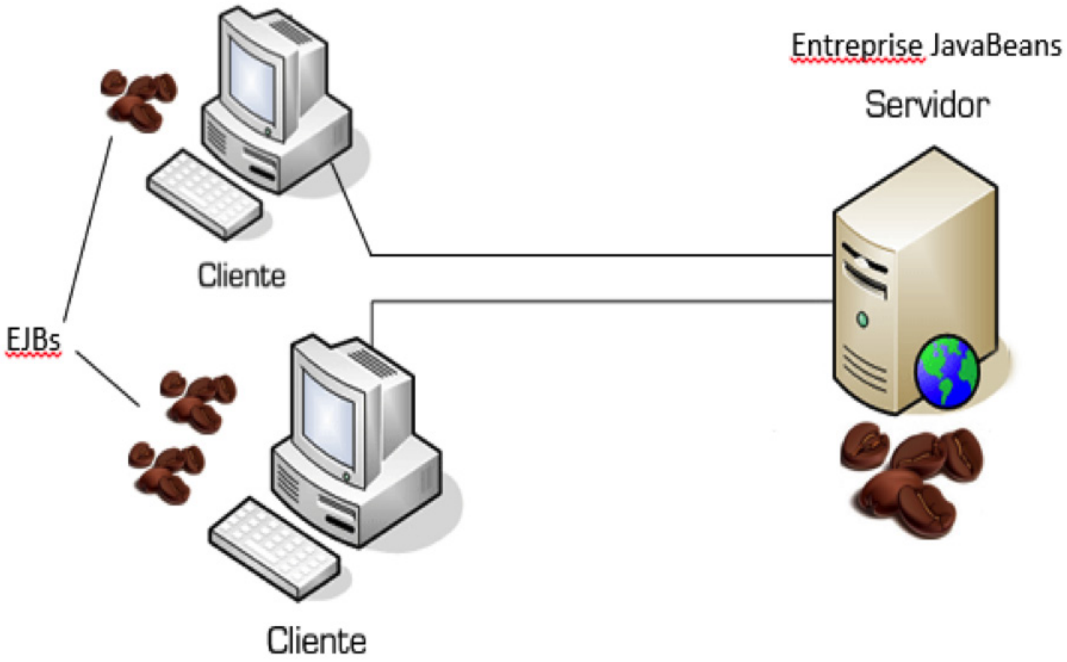


Figura 1. Objetos de Java Beans del lado del servidor y del cliente
Fuente: Propia.

Desde la versión 6 de Java Enterprise Edition, con la integración de JPA, Java Persistence API. Los EJB quedan limitados a los Java Beans Session que trataremos en esta cartilla.

Los Session Beans, se encargan de encapsular la lógica de negocio de las aplicaciones empresariales, permite separar de forma clara para el desarrollo la lógica del negocio entorno a los requerimientos como la escalabilidad, la seguridad, la apariencia y transacciones, etc. Esto se puede aplicar tanto a un equipo de trabajo complejo como también para distribuir de forma racional en trabajo de proyecto individuales.

Los objetos EJB, residen siempre en el servidor y se requiere de un cliente que interactúe con ellos, entre el cliente y el servidor se dan dos tipos de Beans.

- Stateful Session (Sesiones con estado).
- Stateless Session (Sesiones sin estado).

La diferencia radica en que la interacción en los Stateful, no se cierra, esta permanece abierta entre las invocaciones a los métodos que interactúan entre el cliente y el servidor. Mientras que en los Stateless se cierra la sesión después de la invocación de método y hay que volverla a abrir cuando se invoque un nuevo método que necesite llevar a cabo esa iteración con el servidor.

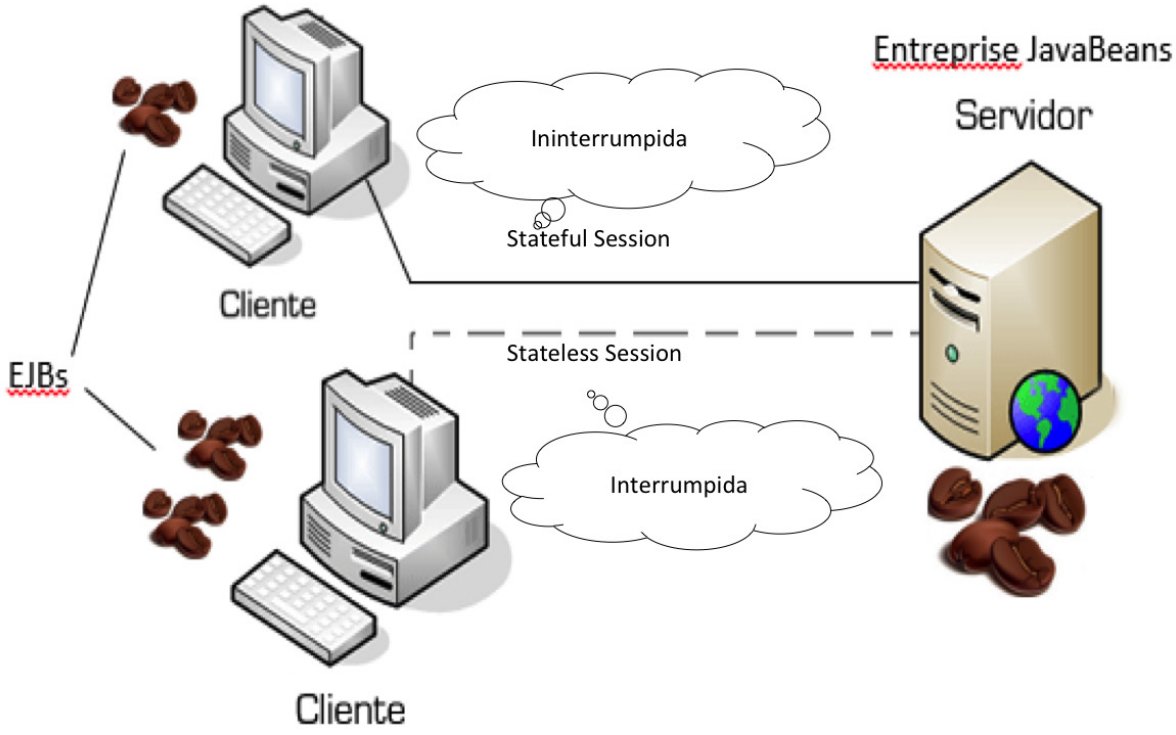


Figura 2. Tipos de sesiones Java Beans
Fuente: Propia.

En Java podemos crear Sesión Beans en tres tipos de proyectos:

- Enterprise Aplication (Pueden contener EJBs y sus clientes).
- EJB Module (sólo pueden contener EJBs).
- Web Application (Integrado con proyecto Enterprise Application, aplicaciones independientes).

Esta posibilidad de añadir EJB a aplicaciones web es una nueva característica que se introduce en Java Enterprise Edition 6.0 y que facilita el empaquetado y el despliegue de aplicaciones web que usen estos Enterprise Java Beans en este caso los Session Beans. Lo que permite empaquetar de forma fácil el código de la aplicación web y el código EJB en un único archivo WAR.

En esta cartilla vamos a trabajar con una aplicación web como cliente, para negociar con los Sessions Beans, podemos también utilizar clientes independientes. Es importante consultar u poco el link www.java.com/es/download/faq/java_webstart.xml. Allí encontrarán mucha información relacionada con el Java Web Start.

¿Qué es Java Web Start y cómo se ejecuta?

¿Qué es Java Web Start?

El software de Java Web Start permite descargar y ejecutar aplicaciones Java desde la Web. Características de Java Web Start.

- Permite activar las aplicaciones con un simple clic.
- Garantiza que se está ejecutando la última versión de la aplicación.
- Elimina complejos procedimientos de instalación o actualización.

Recomiendo visitar el link para estar un poco más empapados del tema que relacionaremos en esta cartilla.

Para iniciar un ejemplo de Session Beans vamos a trabajar con el IDE Netbeans en nuestro ejemplo vamos necesitar 4 proyectos enlazados entre sí, con la siguiente función.

- SessionBean : proyecto Enterprise Application.
- SessionBeanCliente: App de cliente.
- SessionBeanClienteLib: Proyecto de librería de clases.
- SessionBean-ejb : Proyecto para la Sesion Beans.

Vamos crear un proyecto Java Enterprise Edition, para esto damos clic en proyecto nuevo, seleccionamos en categorías Java EE y en proyectos seleccionamos Enterprise Application.

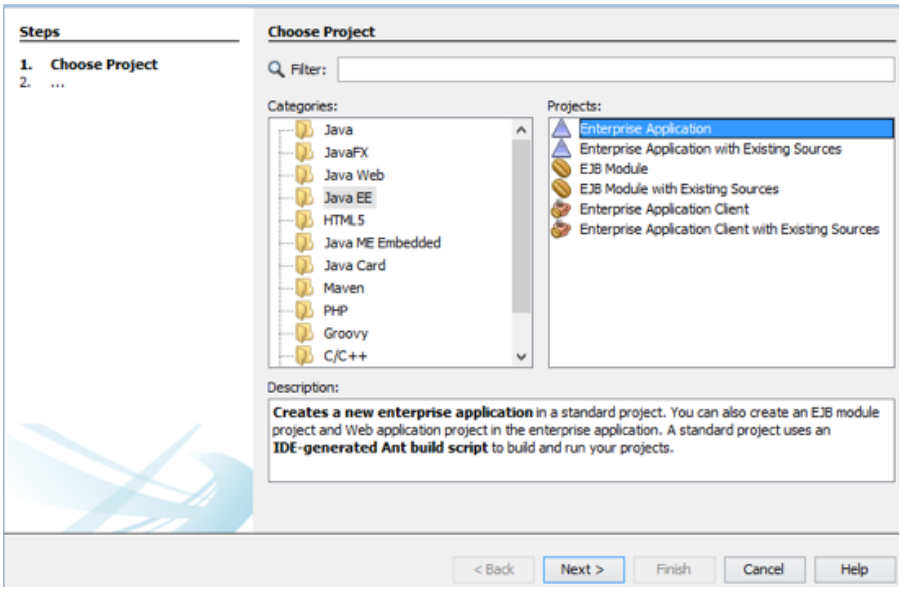


Figura 3. Creación del proyecto Java Beans, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Damos clic en Next>, asignamos el nombre al proyecto "SessionBeanPrimero", y dejamos por defecto, damos clic en Next>.

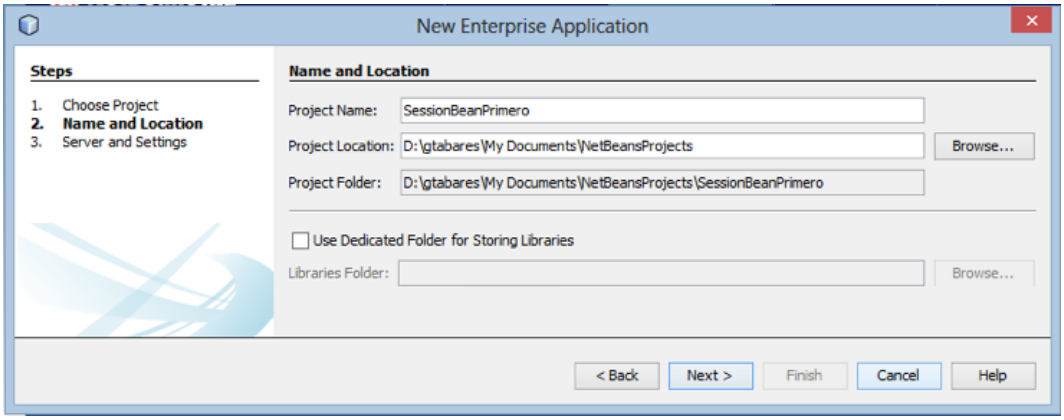


Figura 4. Definiendo nombre del proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

A continuación debemos seleccionar los módulos que deben der incluidos en la aplicación, seleccionamos "Create EJB Module", en este caso no vamos a utilizar "Create Web Application Module", lo desactivamos ya que vamos a utilizar una aplicación Enterprise.

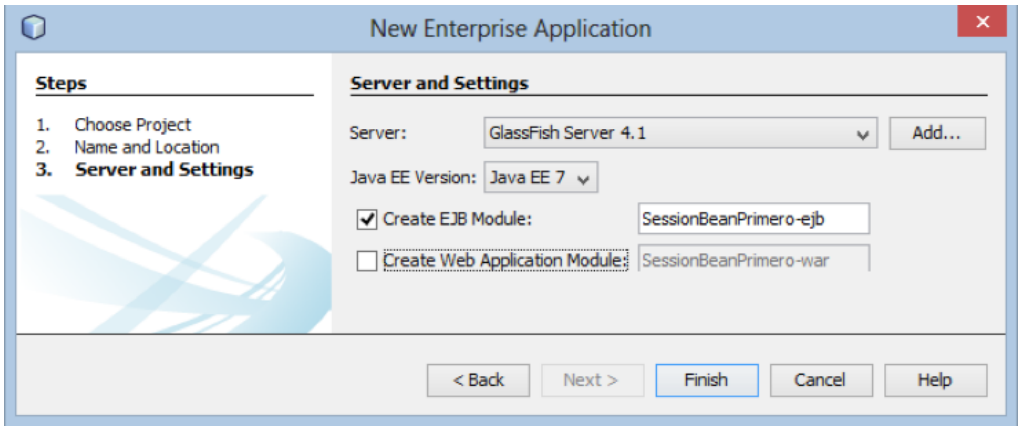


Figura 5. Selección del servidor, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Para finalizar damos clic en Finish, con esto se hemos creado dos proyecto en nuestro ejemplo.

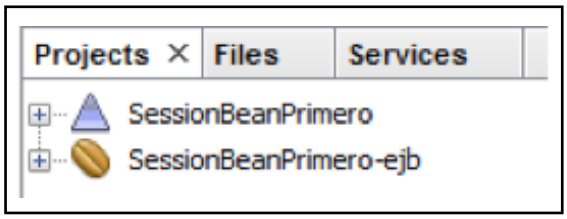


Figura 6. Proyecto Java Beans, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Vamos a crear un cliente, a través de un tercer proyecto que será el módulo de la aplicación cliente.

Vamos crear un proyecto Java Enterprise Edition, para esto damos clic en proyecto nuevo, seleccionamos en categorías Java EE y en proyectos seleccionamos Enterprise Application Client.

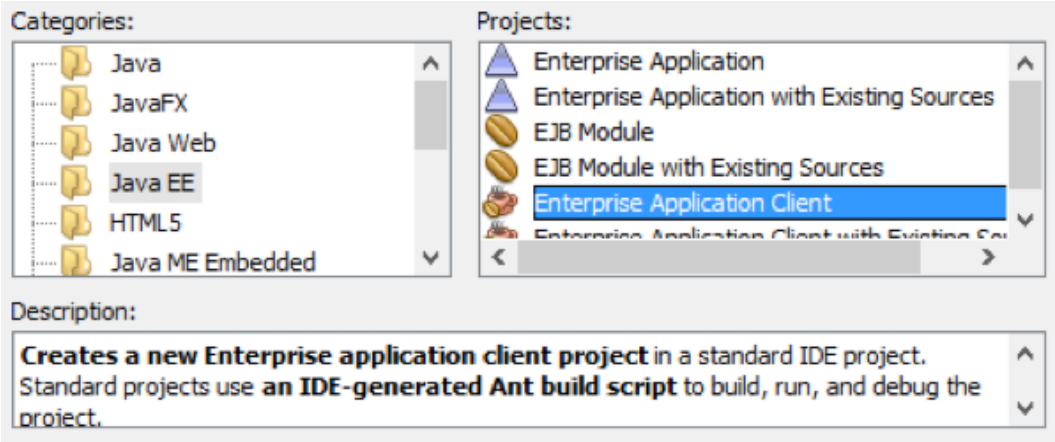


Figura 7. Creación proyecto cliente Enterprise, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Damos click en Next>, y asignamos el nombre al proyecto "SessionBeans_primero_cliente", dejamos el resto por defecto y clic en Next>,

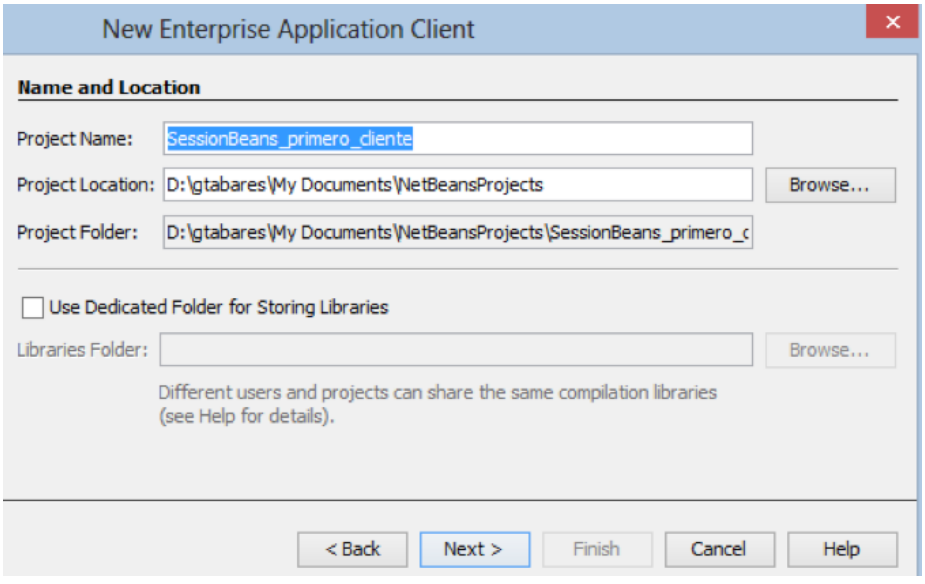


Figura 8. Asignando nombre al proyecto cliente Beans, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

A continuación, añadimos nuestro proyecto a una aplicación Entreprise Application, "Add to Enterprise Application", seleccionamos "SessionBeanPrimero", que es el proyecto Enterprise que hemos creado incialmente, damos clic en Finish.

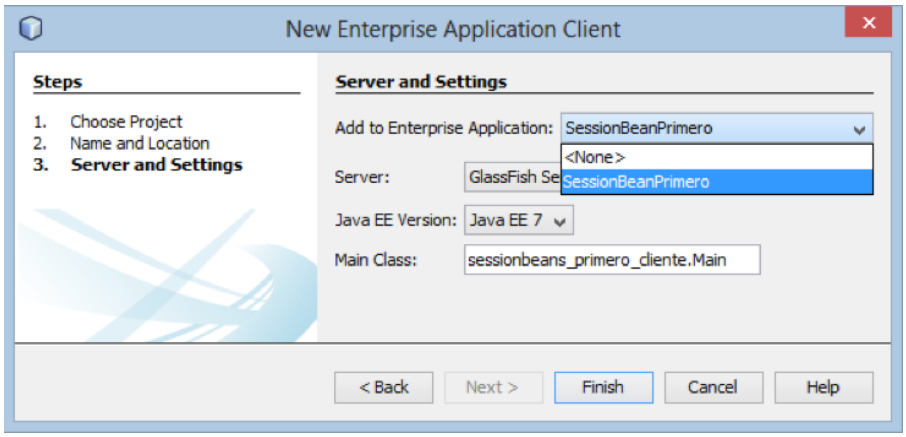


Figura 9. Selección del servidor cliente, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Y tendremos el tercer proyecto "SessionBean_primero_cliente", con la siguiente estructura.

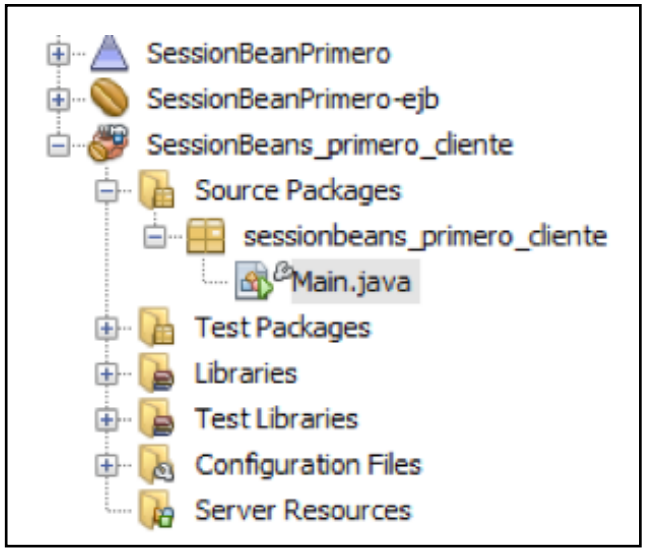


Figura 10. Lista de archivos del proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

En este proyecto tenemos la clase Main.java que utilizaremos más adelante. Ahora crearemos nuestro cuarto proyecto de librerías de clases. Damos clic en proyecto nuevo, seleccionamos en categorías Java y en proyecto seleccionamos Java Class Library.

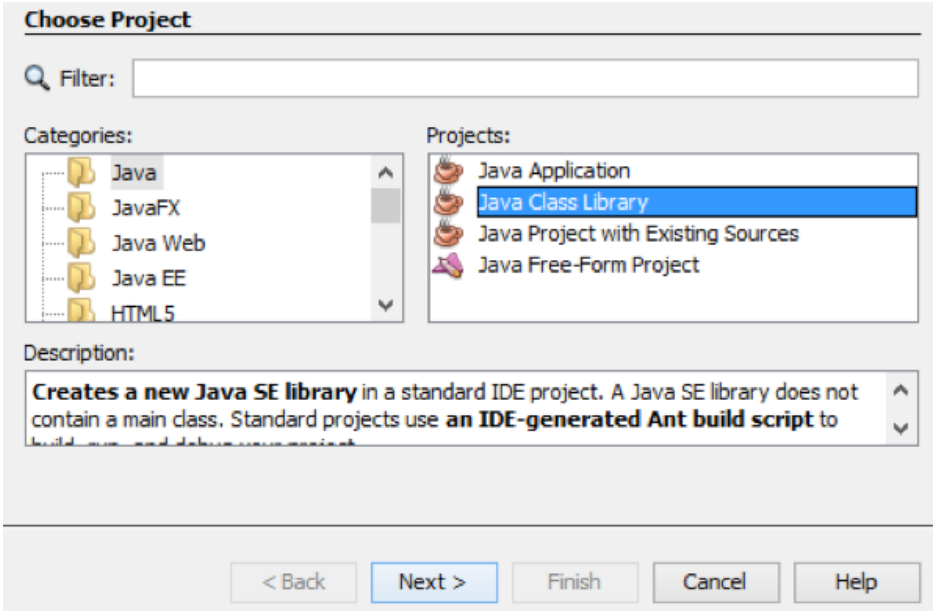


Figura 11. Creando el archivo de librerías del proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Damos clic en Next>, y le asignamos el nombre al proyecto "SessionBeanPrimero_Cliente-Lib", y damos clic en Finish.

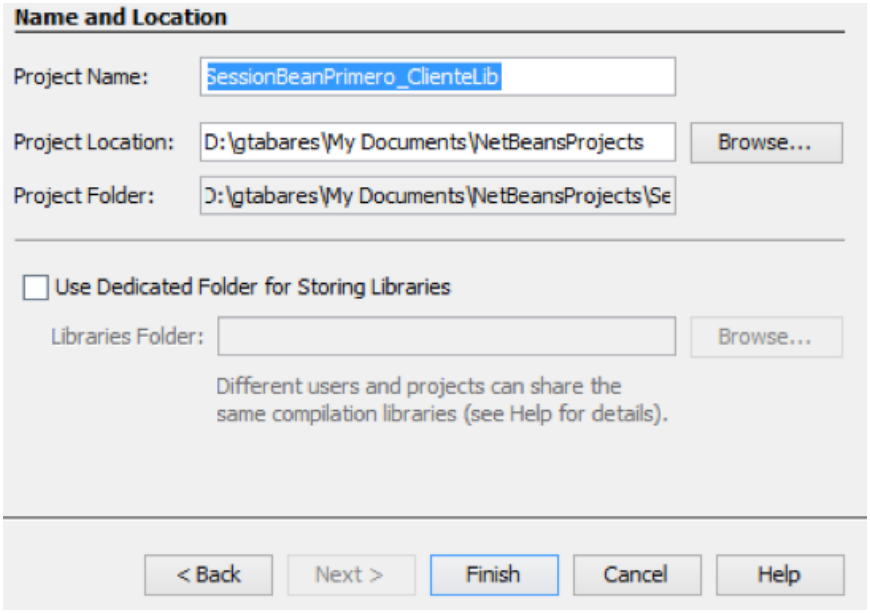


Figura 12. Asignando el nombre del proyecto cliente Lib, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

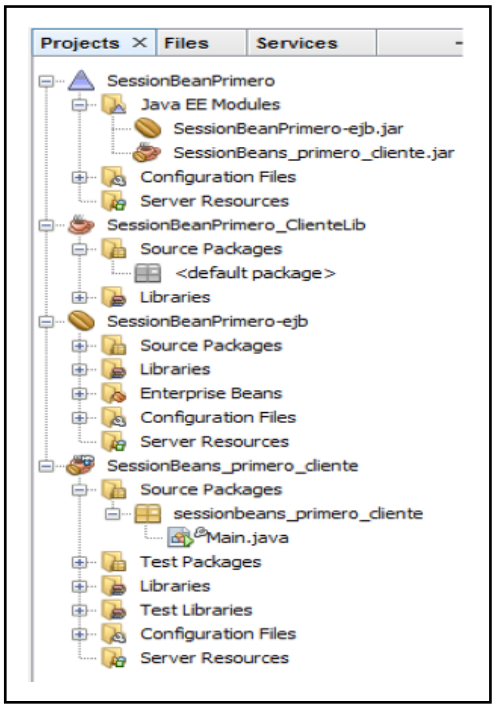


Figura 13. Lista de archivos de todo el proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Y con esto, tenemos nuestro cuarto proyecto creado, el cual debe tener la siguiente estructura.

Contamos con 4 proyectos con los nombres:

- SessionBeanPrimero
- SessionBeanPrimero_ClienteLib
- SessionBeanPrimero-ejb
- SessionBeans_primero_cliente

A continuación vamos a crear nuestro Session Beans, para esto seleccionamos el proyecto "SessionBeanPrimero-ejb" clic botón derecho del mouse y seleccionamos new / Other, en la ventana seleccionaremos en lista de categoría "Enterprise JavaBeans" y en tipo de archivos "Sesión Bean", damos clic en Next>.

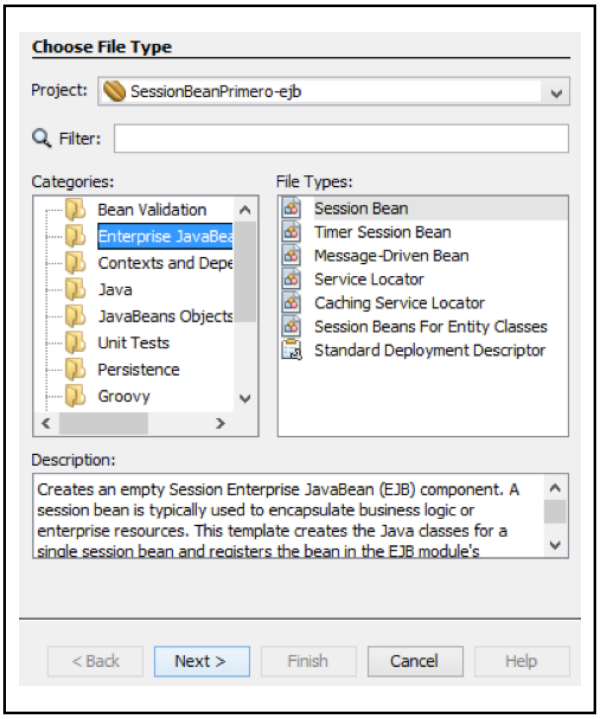


Figura 14. Creación de Session Beans, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Le asignamos un nombre al archivo "CargarSession", creamos un paquete "com.sessionbeanprimero.ejb", seleccionamos el tipo de Session "Stateless", creamos una interfaz "Remote" y seleccionamos el proyecto de "SessionBeanPrimero_ClienteLib" y damos clic en Finish.

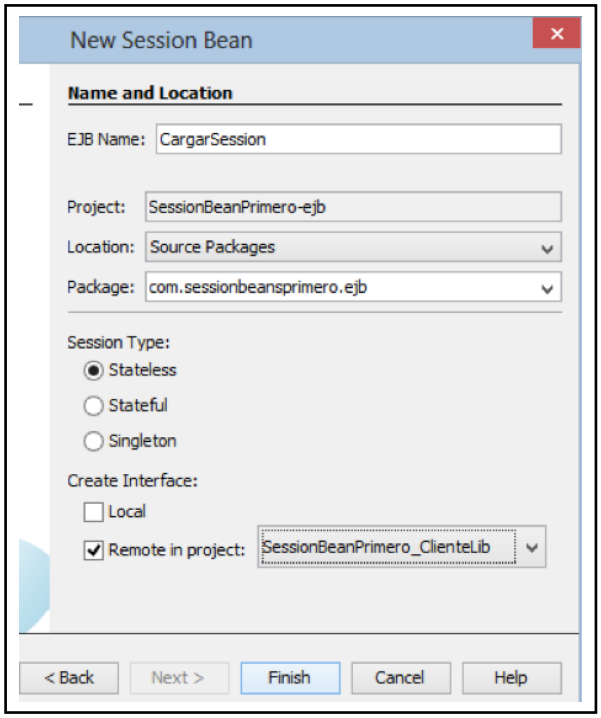


Figura 15. Definiendo el tipo de Session de la aplicación imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Vamos a explicar un poco en las opciones de la figura 15:

Sessions Type:

- Stateless: sesiones sin estado, No se mantiene la comunicación, la aplicación se ejecuta de modo fluido, pero los valores de las variables no serán consistentes.
- Stateful: sesiones con estado, mantiene una comunicación entre el servidor y el cliente, por lo tanto los valores que mantengan las variables miembros son consistentes entre distintas llamadas de los métodos sin perder la información de los datos.
- Singleton: este tipo de Session fue incorporado a partir de la versión Java Entrerprise 6.0 y se encarga de crear instancias únicas de hilo para cada Session Beans, se crea cuando es desplegada nuestra aplicación, trabaja como cache ideal para trabajar con datos que deban ser leídos muy frecuentemente desde la base de datos.

Create Interface: debemos especificar si nuestra aplicación va a tener una interfaz local o remote.

- Local: El cliente se ejecuta en la misma máquina virtual Java donde también se ejecuta el Java Beans.
- Remote in Project: Es usada para clientes que se ejecutan en máquinas virtuales java diferentes a las que se encuentran los Beans de Sessions.

Se puede seleccionar las dos interfaces Local y Remote.

Los proyectos tendrán la siguiente estructura:

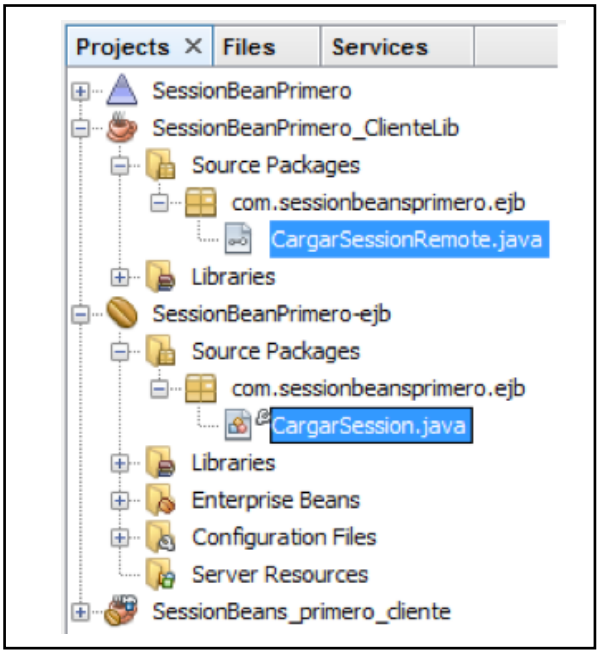


Figura 16. Tipos de acceso a la aplicación, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Al analizar los proyectos SessionBeanPrimero-ejb, donde creamos la clase CargarSession. java, está también fue creada en el proyecto SessionBeanPrimero_ClienteLib, con el mismo nombre del paquete y con el nombre CargarSessionRemote.java que es nuestro cliente remoto.

La clase CargarSession.java contiene la notación @Stateless que es el tipo de sesión que hemos seleccionado. La clase implementa la clase "CargarSessionRemote" que es nuestro cliente remoto.

```
package com.sessionbeansprimero.ejb;

import javax.ejb.Stateless;

/**
 *
 * @author gtabares
 */
@Stateless
public class CargarSession implements CargarSessionRemote {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

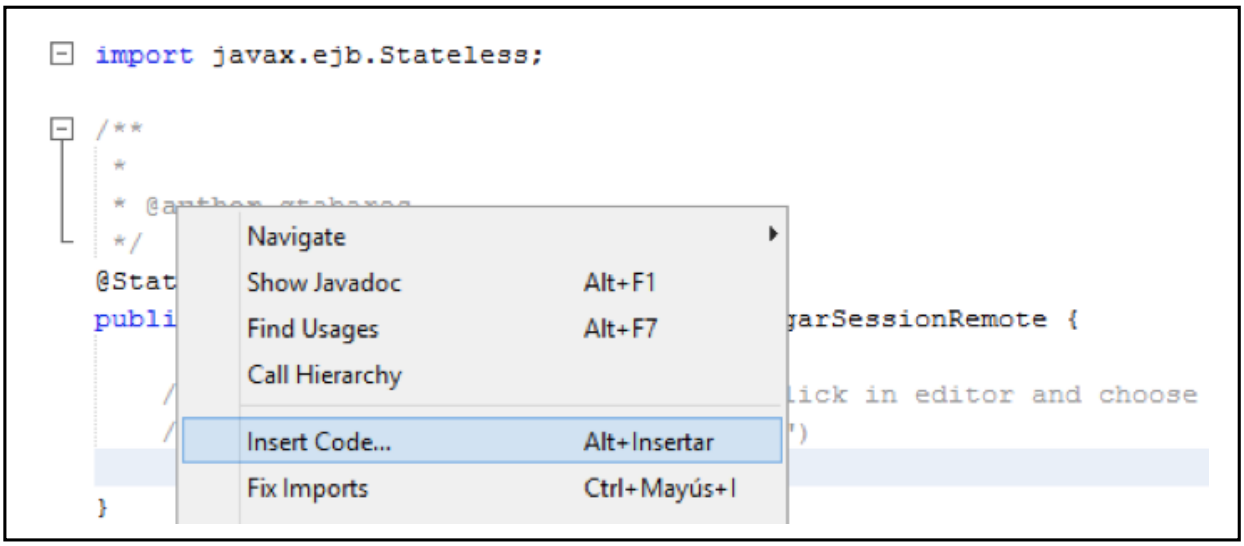
Veamos la estructura de la clase CargarSessionRemote.java que se creó automáticamente con la Interfaces que seleccionamos "Remote".

```
package com.sessionbeansprimero.ejb;

import javax.ejb.Remote;

/**
 *
 * @author gtabares
 */
@Remote
public interface CargarSessionRemote {

}
```

Para insertar código en nuestra aplicación, vamos a la clase CargarSession.java, y en la clase nos indica:

```
// Add business logic below. (Right-click in editor and choose
// "Insert Code > Add Business Method")
```

Adicionar la lógica del negocio abajo, con el botón derecho del mouse en el editor y seleccionar "Insert Code" y posteriormente Add Bussiness Method, con esto el método que vamos a crear también se creará en la clase CargarSessionRemote.java. Vamos a hacerlo.



Seleccionamos "Add Bussiness Method". Nos cargará una ventana para asignar el nombre al método, le colocamos "cargar" y colocamos el tipo de retorno del método en este caso escribimos "String" y adicionamos los parámetros de nuestro método, en este caso adicionamos uno llamado cargando, como muestra la figura.

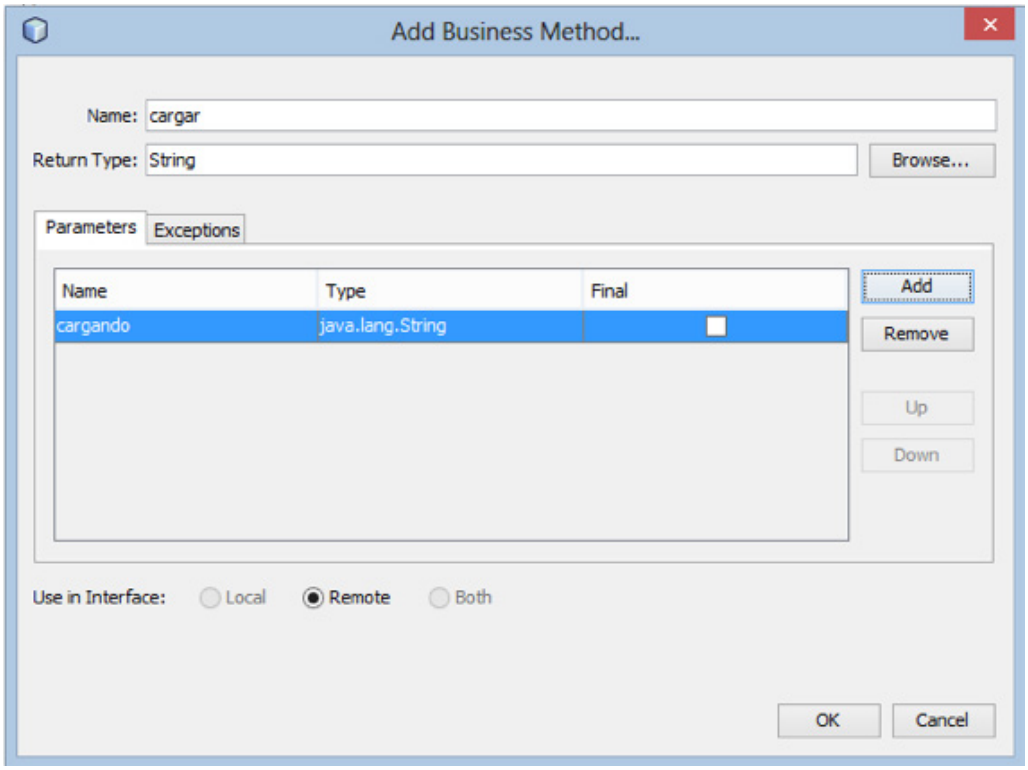


Figura 17. Definición del método del negocio, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Con esto lo que hicimos fue crear un método de tipo String llamado cargar, retornando un String por defecto. Vamos guardando los cambios del proyecto.

Si observamos las dos clases a las que hemos hecho referencia CargarSession.java y CargarSessionRemote.java, se ha creado el método en la clase CargarSessionRemote.java automáticamente.

```
@Remote
public interface CargarSessionRemote {

    String cargar(String cargando);

}
```

Y la clase CargarSession.java, la cual vamos modificar para que retorne un valor, agregando el siguiente código.

```
package com.sessionbeansprimero.ejb;

import javax.ejb.Stateless;
/**
 *
 * @author gtabares
 */
@Stateless
public class CargarSession implements CargarSessionRemote {

    @Override
    public String cargar(String cargando) {
        return "Cargado: "+cargando;
    }
}
```

Hasta aquí ya tenemos un Session Beans que es accedido por un cliente de forma remota y devuelve un String.

Ahora nos centraremos en el proyecto SessionBean_primero_cliente el cual debe usar la librería que se ha creado en el proyecto SessionBeanPrimero_ClienteLib

Desplegamos el proyecto SessionBean_primero_cliente, pulsamos sobre libraries con el botón derecho del mouse y seleccionamos Add Project…

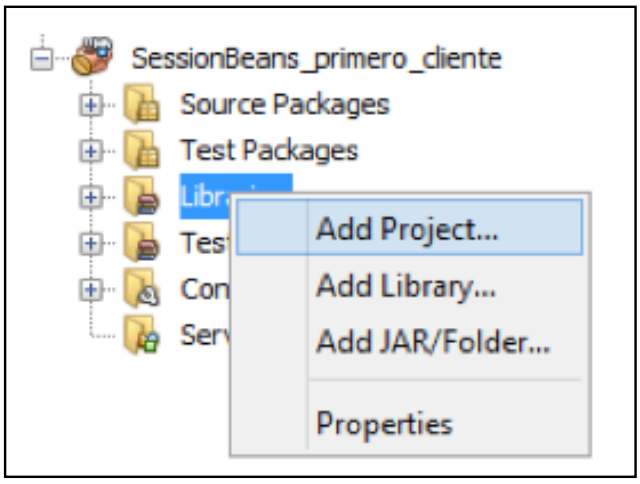


Figura 18 Asignando la librería al proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Y buscamos en la lista de proyectos "SessionBeanPrimero_ClienteLib", donde está la sesión remote y clic en Add Project JAR Files.

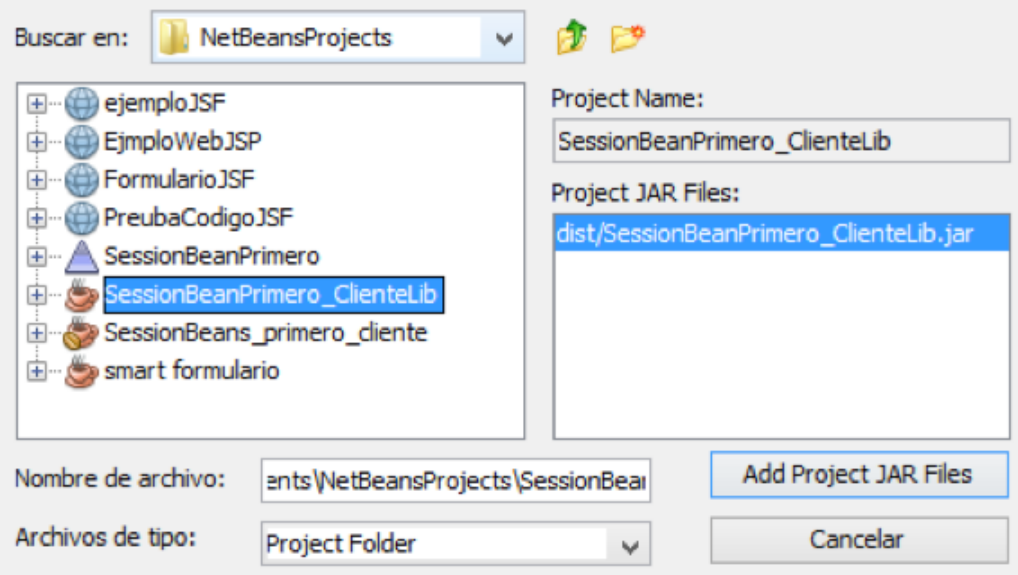


Figura 19. Seleccionando la librería para el proyecto, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ahora es necesario invocar desde el cliente el método que está en la SessiónBeans, para esto es necesario crear una instancia o referencia de la clase que implementa la interfaces remota, nos ubicamos en el proyecto que estamos desarrollando "SessionBeans_primero_cliente", abrimos el archivo Main.java pulsamos con el botón derecho del mouse y seleccionamos insert código, seleccionamos "Call Enterprise Bean"…

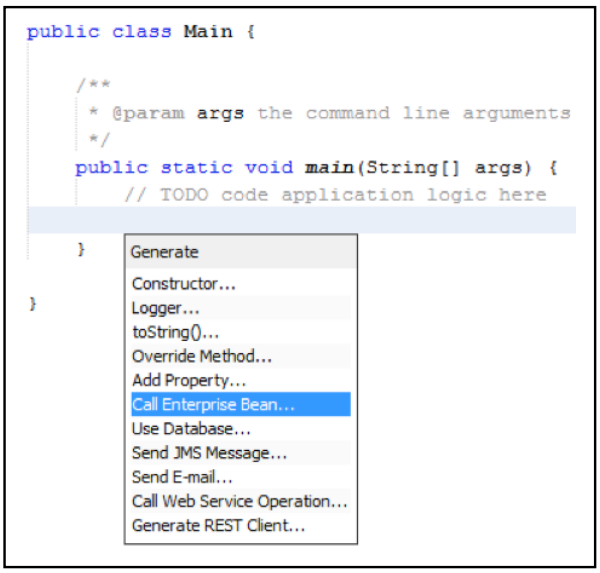


Figura 20. Llamada al Bean, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos un Enterprise bean de la lista de Bean crados

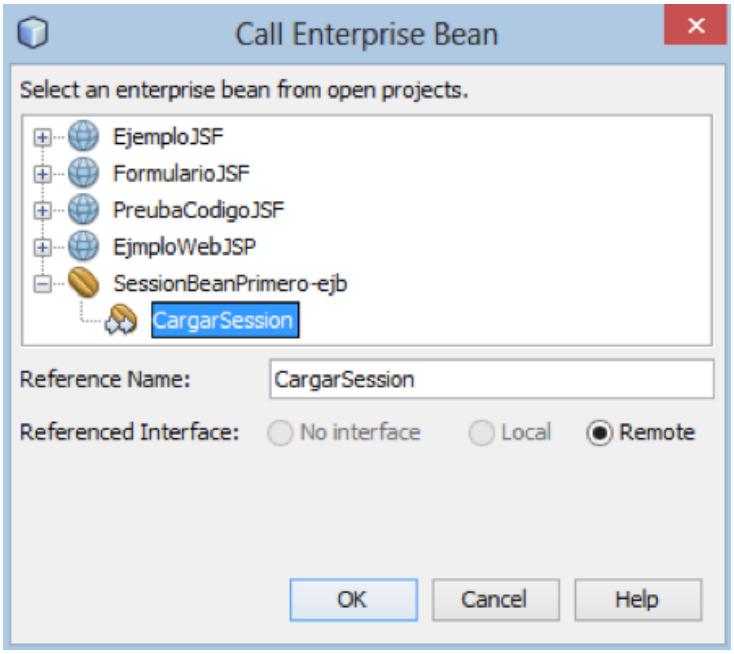


Figura 21. Llamada al Beans cargarSession, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ha insertado en el código la notación @EJB y ha importado las librerías necesarias para la clase, creando una variable "cargarSession" de tipo CargarSessionRemote. Como lo muestra la figura 21.

```
package sessionbeans_primero_cliente;

import com.sessionbeansprimero.ejb.CargarSessionRemote;
import javax.ejb.EJB;

/**
 *
 * @author gtabares
 */
public class Main {
    @EJB
    private static CargarSessionRemote cargarSession;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here

    }

}
```

Agregamos el código haciendo uso la libraría que adicionamos al proyecto, llamando al método cargar del SessionBeanPrimero_ClienteLib.

cargarSession.cargar(«Mostrando el metodo de la libraria a traves de la Session Remota»);

```
public static void main(String[] args) {
    // TODO code application logic here
    cargarSession.cargar("Mostrando el metodo de la libraria a traves de la Session Remota");
}
```

Ejecutamos el proyecto SessionBeanPrimero.

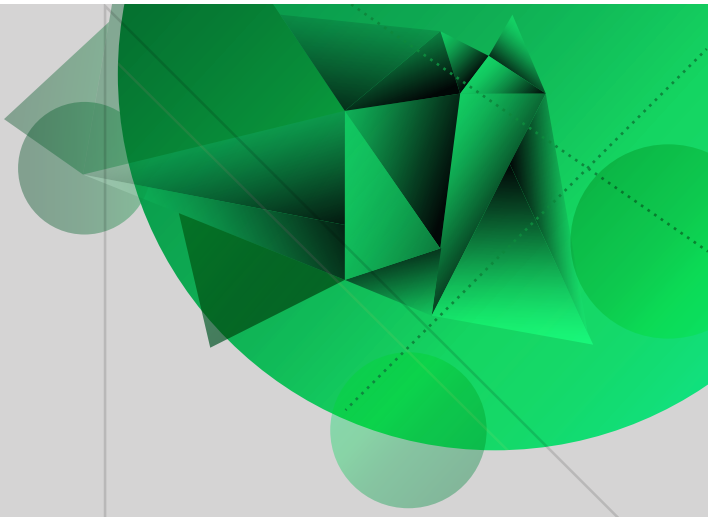
Para ejecutar el proyecto Enterprise.

```
Cargado: Mostrando el metodo de la libraria a traves de la Session Remota
run:
BUILD SUCCESSFUL (total time: 29 seconds)
```

3

Unidad 3

Java persistence API JPA



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Introducción a Java Persistence API, más conocida por sus siglas JPA es el API de persistencia desarrollada por la plataforma JavaEE.

Proporcionar un estándar para gestionar datos relacionales en aplicaciones JSE o JEE, de forma que además simplifica el desarrollo de la persistencia de datos, JPA se encarga de mapear una clase a una tabla de una base de datos, de esta manera solo tenemos que escribir el código de nuestra clase con sus atributos y sus métodos. JPA se encarga de realizar la persistencia de los objetos de dicha clase en la base de datos.

U3 Metodología

Se recomienda seguir en un orden secuencial la lectura y estar muy atento al código que se utiliza para el desarrollo de la unidad. En cada ejercicio que se expone, en la unidad, encontrará la explicación detallada para una mayor comprensión del tema.

Se recomienda realizar ejercicio basados en los ejemplos que están desarrollados en la unidad para comprender mucho más fácil el concepto de interface en Java.

Java persistence API JPA

Java Persistence API, JPA permite la persistencia no solo en la base de datos, sino que también en otros formatos como archivos de texto plano y XML. Las aplicaciones empresariales necesitan recolectar, procesar, transformar y reportar grandes cantidades de información, toda esta información debe estar almacenada en algún lugar. Java ofrece una solución estándar denominada Java Persistence API o JPA que facilita el trabajo al desarrollador de software ya que sirve de puente entre el mundo de desarrollo y el mundo de las bases de datos, permitiendo una mayor gestión de la información que es crucial al mundo de hoy para muchos negocios.

Mencionemos las diferencias entre el modelo orientado a objetos y el modelo relacional, los datos manejados por algún sistema en algún momento terminaran siendo almacenados en Bases de Datos para su posterior recuperación y análisis. Las bases de datos relacionales almacenan los datos en tablas formadas por filas y columnas.

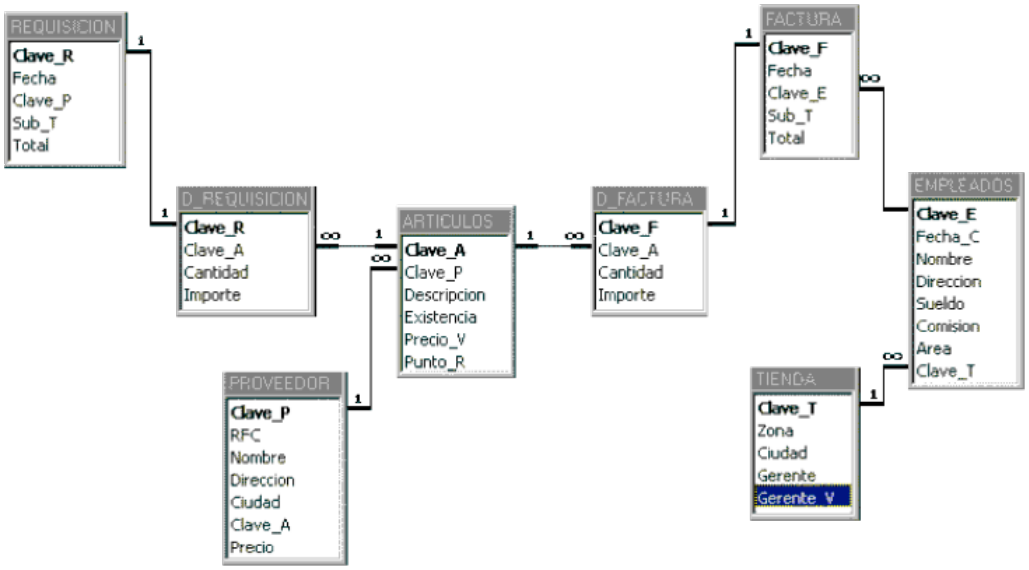


Figura 1. Modelo Relacional
Fuente: http://www.esandra.com/mysql-i-introduccion-a-las-bases-de-datos-relacionales/

Cada registro usa como identificar un único campo de llaves primarias, las relaciones entre tablas se representan mediante claves foráneas o tablas joins, respetando restricciones de integridad, toda esta terminología es propia del modelo relacional. Sin embargo es totalmente desconocido en el modelo orientado a objetos a objetos propio de Java.

En la orientación a objetos se manejan clases y sus instancias denominadas objetos; los objetos encapsulan su estado y comportamiento mientras está corriendo la máquina virtual de Java, si se detiene la máquina se caen o se pierden todos los datos del negocio que maneja el sistema en ese momento, para evitar esto necesitamos mecanismos de persistencia.

La persistencia es el proceso de almacenar el estado de un objeto de forma permanente en disco duro, el proceso consiste en trasladar los datos de un modelo orientado a objetos a otro relacional. En Java existen tres formas para logar persistencia.

Métodos para lograr la persistencia

- Serialización: Java.io.Serializable: que convierte los objetos en una serie de bit almacenable en disco. Tiene la desventaja de que carece de lenguaje de consulta y no da soporte a altas tasas de acceso concurrente.
- Java Data Connectivity: JDBC: Permite acceder a sistemas gestores de Bases de datos relacionales, aunque su uso está muy extendido, la tendencia es migrar a herramientas más potentes de mapeo de objetos relacional, esta herramientas se base en delegar el acceso a la base de datos relacionales a un framework, Java define el modelo de programación para la persistencia de entidades, pero deja su implementación a proveedores externos, entre los que tenemos IBM con Hibernate.
- Herramientas de mapeo Objeto-Relacional (ORM) JPA: Se encarga de automatizar la asignación de objetos Java a tablas de bases de datos relacionales, sustituyen a las entitys Beasn.

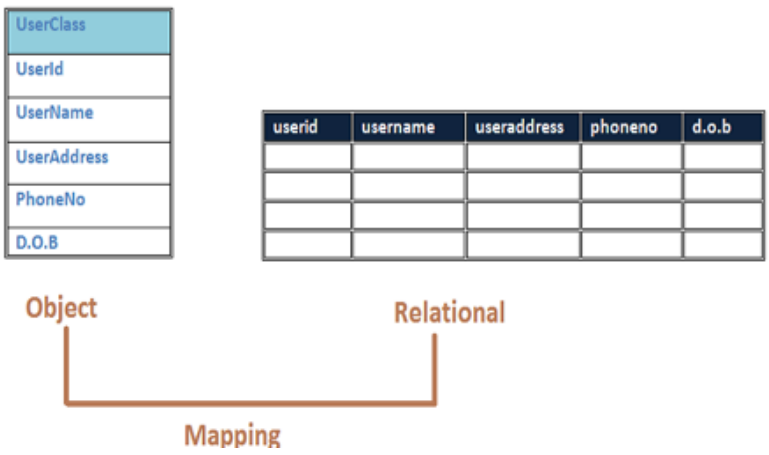


Figura 2. Mapeo Relacional
Fuente:http://blog.jhades.org/solving-orm-complexity-keep-the-o-drop-the-r-no-need-for-the-m/

JPA trabaja a través de entities que son clases Java también llamadas POJOS (Plain Old Java Objects), no necesitan extender a ninguna otra clase o interface y se definen a través de anotaciones en código que desarrollemos.

```
@Entity
@Table(name = "CUSTOMER",schema = "APP")
public class Customer implements Serializable {
    @Id
    @Column(name = "CUSTOMER_ID")
    private Integer customerId;
    @Basic(optional = false)
    @Column(name = "DISCOUNT_CODE")
    private char discountCode;
    @Basic(optional = false)
    @Column(name = "ZIP", length = 10)
    private String zip;
    @Column(name = "NAME", length = 30)
    private String name;
```

@entity: Sirve para indicar que una clase Java es una entidad JPA.

Los proyectos que utilizan JPA requieren de una unidad de persistencia, entre las que permite trabajar en el IDE Netbeans utilizaremos eclipse link por defecto.

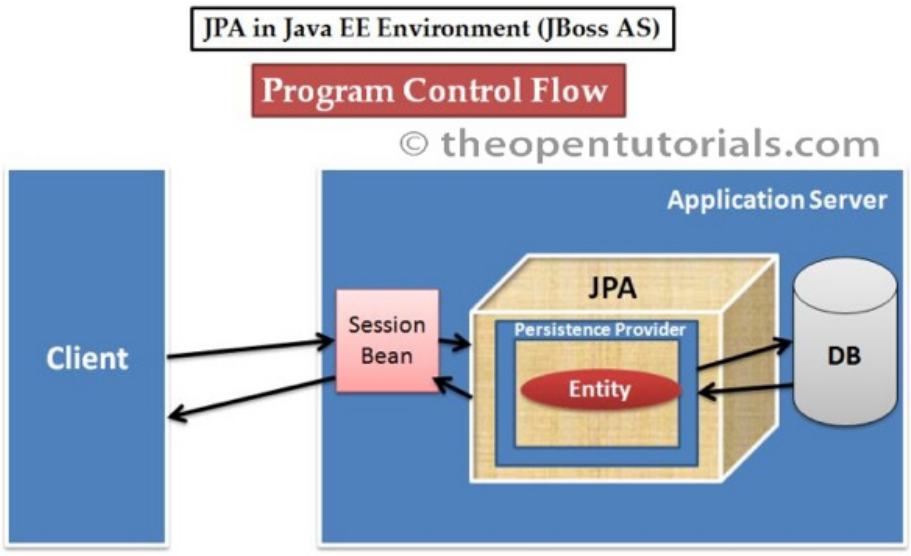


Figura 3. Flujo de control del programa, Imagen tomada del sitio
Fuente: http://2.bp.blogspot.com/-aBGrV32c0ZE/VDYyN6aDceI/AAAAAAAAizA/Q_xcNr28g44/s1600/jpa-in-Javaee-env-1.jpg

Figura 4. Creación de una aplicación web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

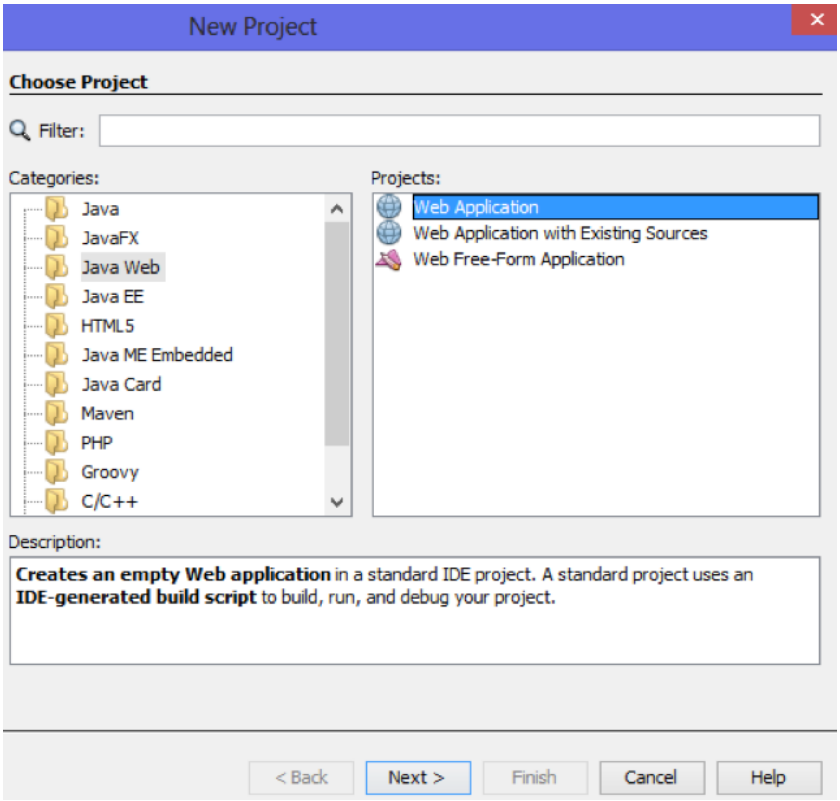
Persistir consiste en guardar datos de un objeto Java en una base datos, a través de conectar una base datos mediante una conexión pool y una fuente de datos.

Java define en un modelo de programación de persistencia por el cual, solo es necesario proporcionarle al motor de persistencia la clase entidad por cada clase de la tabla de la base de datos.

Entidad son objetos que contienen información de mapeo que viven a corto plazo en memora y persistentemente en base de datos, Java Persistence API dispone de una interfaz llamada entity manager, con una serie de métodos para que podamos operar con ese conjunto de entidades. Al conjunto de entidades asociadas con los registros de una base de datos se le llama contexto de persistencia, es importante resaltar que la aplicación lo que maneja son entidades del contexto de persistencia, cualquier cambio se realice en estas entidades se trasladara automáticamente a la base de datos subyacente de forma transparente para nuestro código.

Iniciaremos mostrando un ejemplo para la creación de entidades JPA

Inicialmente se debe crear un proyecto de aplicación Web, con el nombre "ProyectoJAP"



Clic en Next> y seleccionamos como frameworks JavaServer Faces

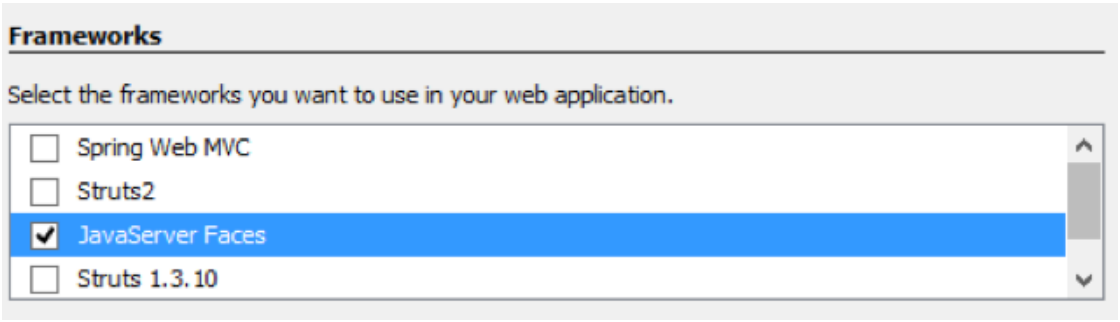


Figura 5. Java Server Faces, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Dejamos por defecto las demás opciones y damos clic en Finish.

Para crear una entity JPA creamos un nuevo archivo y buscamos dentro de la categoría la opción de "Persistencia" y tipo de archivo "Entity Class".

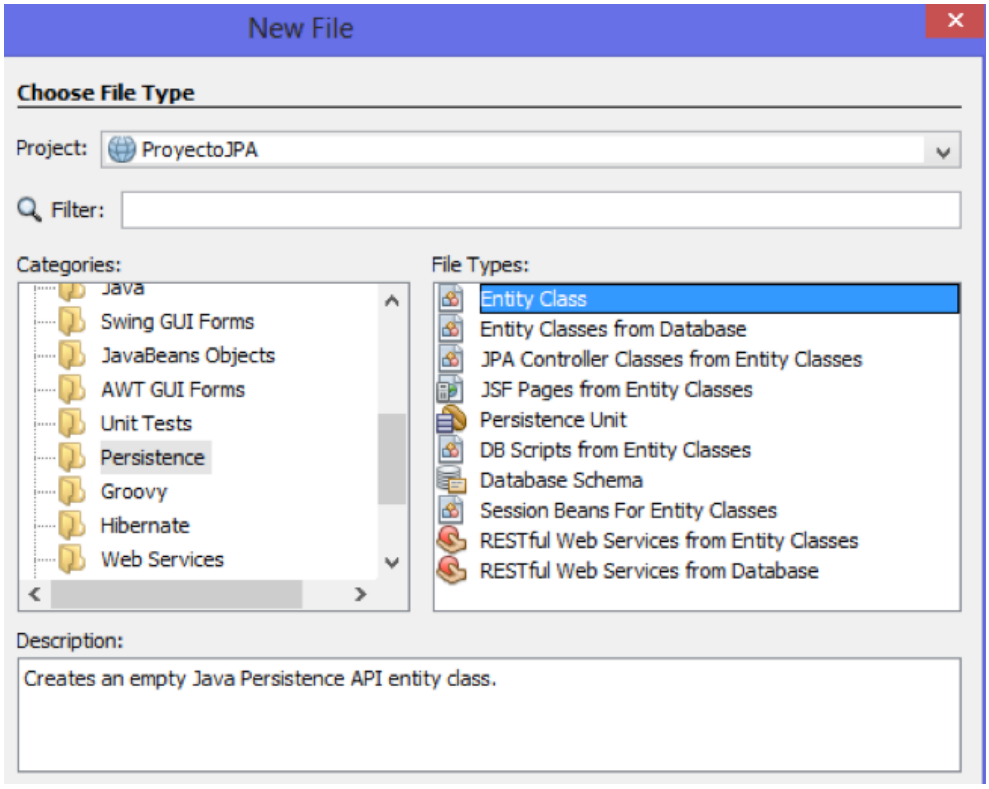


Figura 6. Creación de la Entity Class, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Clic en botón Next>, asignamos nombre a la clase "Estudiante", y creamos un paquete llamado com.persistencia, verificamos que este seleccionado la opción "Create Persistence Unit", parra cada entity class se crea una unidad de persistencia bajo un archivo xmly damos clic en Next>.

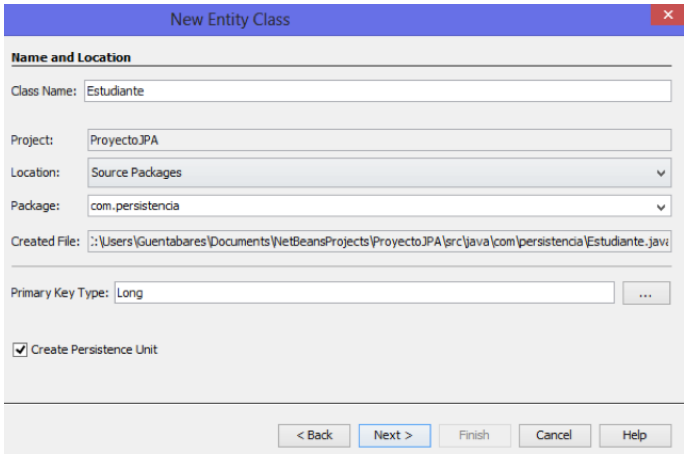


Figura 7. Nueva Entity Class, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Asignamos un nombre a Persistence Unit Name o dejamos el que nos da por defecto, en la opción Persistence Provider seleccionamos el proveedor "EclipseLink (JPA 2.1)(default)" Elegimos la fuente de datos "Data Source".

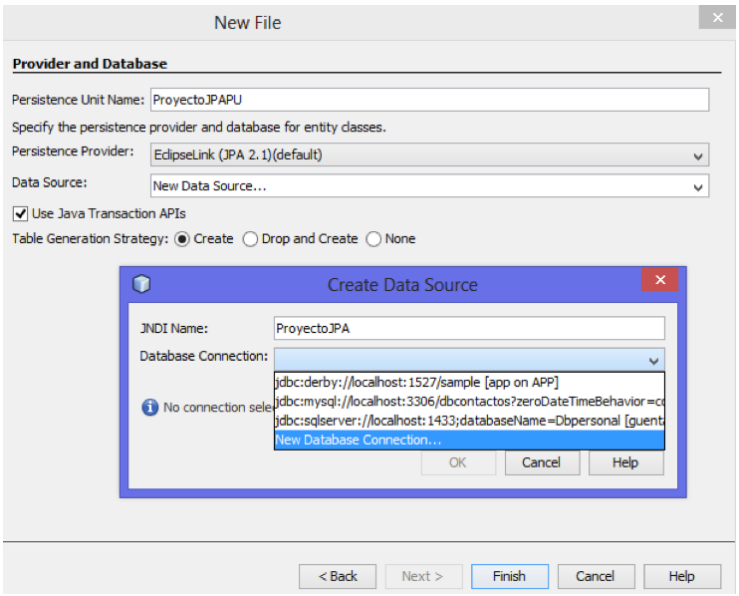


Figura 8. Definición conexión a la Base de Datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Data Source: JPA permite la conexión a la base de datos a través de una conexión pool Java, agrupamiento de conexiones. Estas conexiones nunca se cierran ventaja que tiene frente a JDBC y permiten tener la información del servidor, nombre de la Base de datos, el usuario y clave de acceso a la base de datos.

Para esto en necesario utilizar una instancia de JEE de Javax.sql, Interface DataSource a través de la API (JNDI) Java Naming and Directory.

La interface DataSource, utilizaremos el método getConnection, el cual permite tener una conexión a la base de datos.

Crearemos nuestra propia conexión a la base de datos.

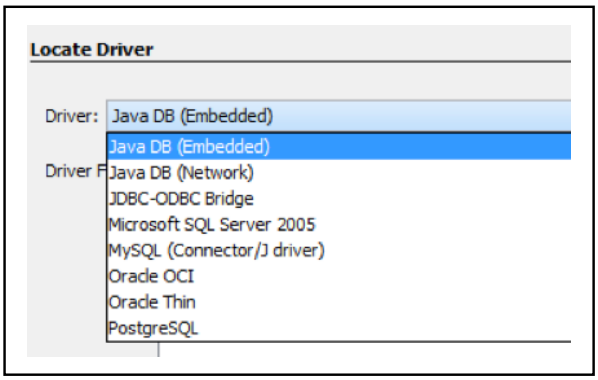


Figura 9.Definición del Driver de la BD, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Aquí encontraremos los Drivers para diferente fuente de datos entre los que encontramos Microsoft, MySQL, Ocracle OCI, PostgreSQL, etc. Para este ejemplo elegimos "Java DB (Network)", que significa Java Derby, se despliegan los driver derbyclient.jar y el derby.jar necesarios para este ejemplo, damos clic en Next>. Proporcionamos los datos de conexión, nombre y usuario a la base de datos.

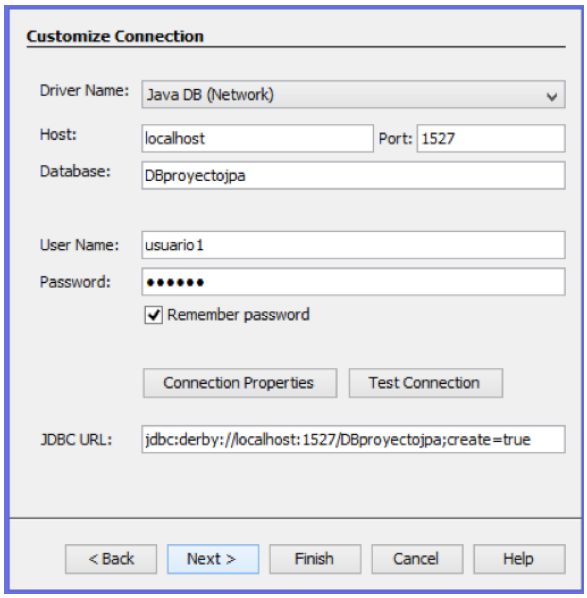


Figura 10. Parámetros de la conexión a la BD, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

En la opción JDBC URL: agregamos a la línea jdbc:derby://localhost:1527/DBproyectojpa; la instrucción de créate=true debido a que la BD no existen para que Netbeans la cree, podemos probar la conexión en el botón "Test Connection". Damos clic en Next>.

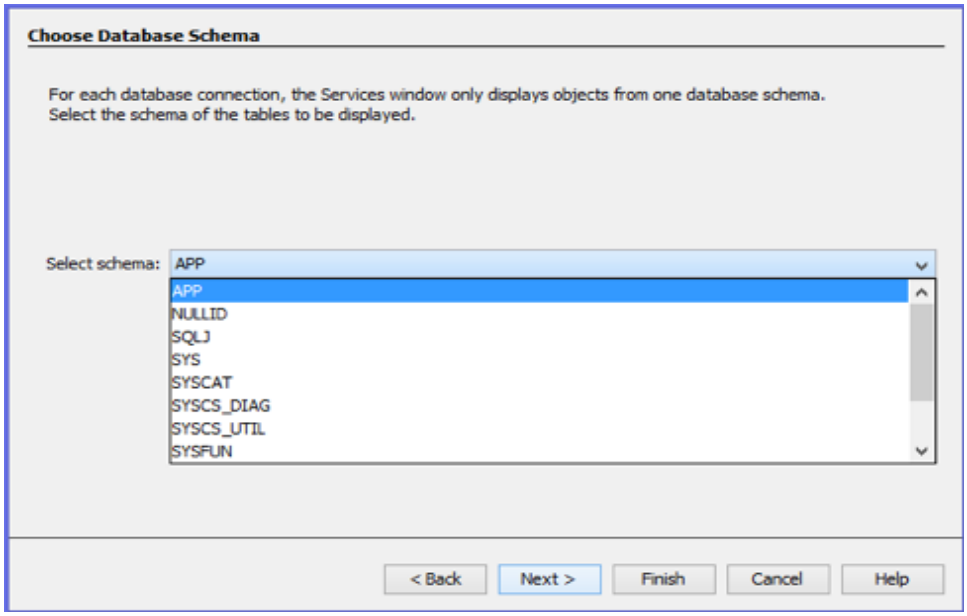


Figura 11. Selección del esquema de la BD, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos el esquema de la DB, por defecto aparece seleccionada APP.

Con esto damos clic en Next>, donde muestra al input de conection a la base de dato, damos clic en Finish, luego en botón Ok.

En la opción Table Generation Strategy, seleccionamos Drop and Create lo que permite cada vez que ejecutemos la aplicación borrar y crear la tabla de la base de datos, pulsamos en el botón Finish.

A continuación se muestra el código generado por Netbeans, para la entity la cual corresponde al archivo Estudainte.Java. Con la anotación @Entity lo que significa que es una clase entidad JPA.

```java
package com.persistencia;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author Guentabares
 */
@Entity
public class Estudiante implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Estudiante)) {
            return false;
        }
        Estudiante other = (Estudiante) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "com.persistencia.Estudiante[ id=" + id + " ]";
    }
}
```

@Id, es indicada para el campo de acceso en JPA es la llave primaria @Id, es indicada para el campo de acceso en JPA es la llave primaria

@GenerateValue, permite generar automáticamente la llave primaria JPA, mediante la estrategia GenetationType.AUTO

Figura 12. Anotaciones de la clave primaria, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

En el siguiente cuadro se muestra las diferentes opciones que se pueden utilizar para cada GenetationType.AUTO.

Enum Constant Summary
AUTO Indicates that the persistence provider should pick an appropriate strategy for the particular database.
IDENTITY Indicates that the persistence provider must assign primary keys for the entity using a database identity column.
SEQUENCE Indicates that the persistence provider must assign primary keys for the entity using a database sequence.
TABLE Indicates that the persistence provider must assign primary keys for the entity using an underlying database table to ensure uniqueness.

Figura 13. Tipos de anotaciones de la entitys, imagen generada por el autor desde el IDE Netbeans
Fuente: Extraída de docs.oracle.com/Javaee/6/api opción GenarationType.

En el archivo Estudiante.Java vamos adicionar los campos nombre y apellido de forma private para la clase entity.

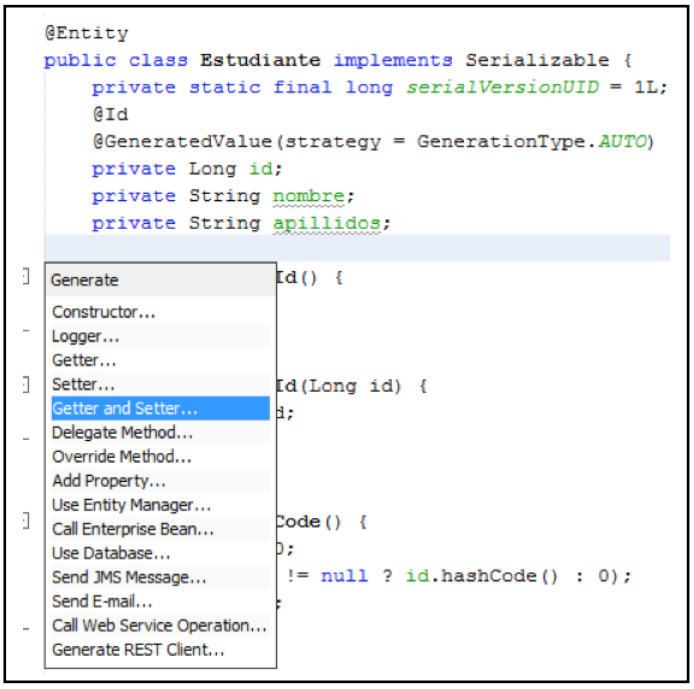


Figura 14. Generación de los métodos Getter y Setter, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Presionamos la tecla Alt + Insert, para crear los métodos Getter and Setter, de la clase.

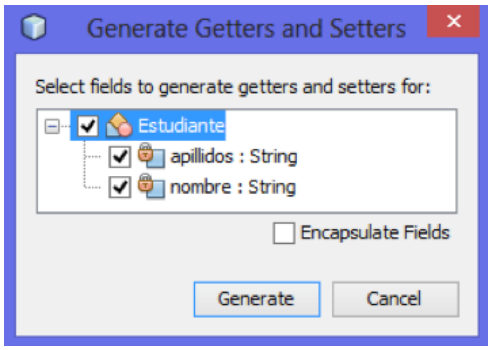


Figura 15. Selección de atributos de la clase, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos los campos y damos clic en Generar, dándonos como resultado los métodos dentro de la clase Estudiante.Java

Data Acess Object DAO

Cuando se desarrollan aplicaciones en Java JEE, en al algunas es necesario acceder a los datos, a través de algún método, por persistencia (Hibernate, JDO, iBatis), JDBC, un Fichero de Texto o LDAP, etc. Cuando tenemos varias opciones para acceder a los datos puede ser un problema, la forma de acceder a los datos muchas veces depende del fabricante y a la forma de almacenamiento que utilicemos.

El Data Acess Object o patrón de diseño DAO, permite resolver este problema abstrayendo y encapsulando el acceso a datos. El DAO se encarga de manejar la conexión con la fuente de datos permitiendo acezar obteniendo y guardando de forma persistente los datos.

A través del DAO este realiza operaciones de forma automáticas hacia la BD, no son necesarias las transacciones, como ejemplos podemos mencionar las búsquedas por ID, los procesos de creación, actualizado y borrado de registros, consultas u otras operaciones que se realicen a menudo en la aplicación.

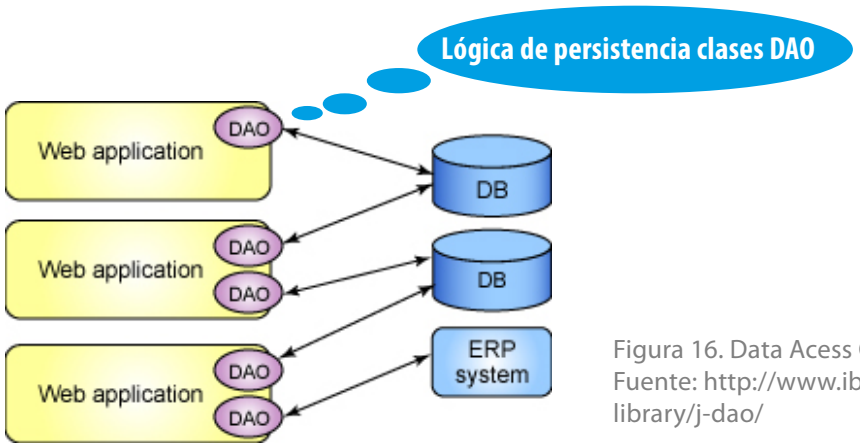


Figura 16. Data Acess Object (DAO)
Fuente: http://www.ibm.com/developerworks/library/j-dao/

Es necesario crear la clase controller JPA de la entity creada anteriormente (Estudiante.Java), las clases controller siguen el patrón de diseño DAO.

Para crear la clase damos clic en archivo, Nuevo Archivo y seleccionamos en la categoría Persistence y en tipos de archivos seleccionamos JPA Controller Classes from Entity Clases y clic en Next>.

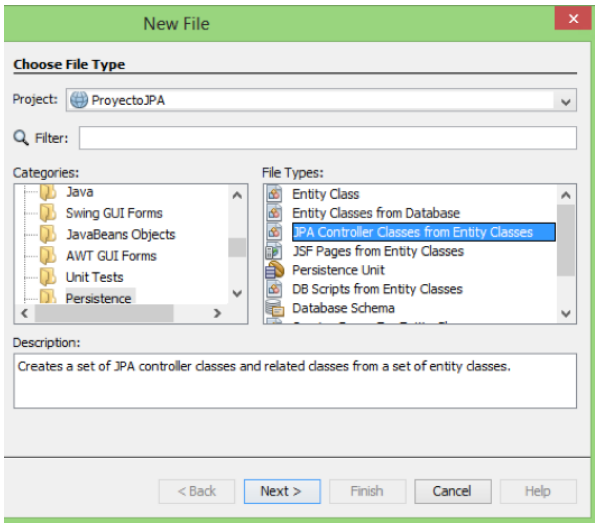


Figura 17. Creación del archivo Clase Controller de la Entity Clase, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Seleccionamos la Entity a la cual le vamos a crear la clase controller y clic en botón add>, luego clic en botón Netx>.

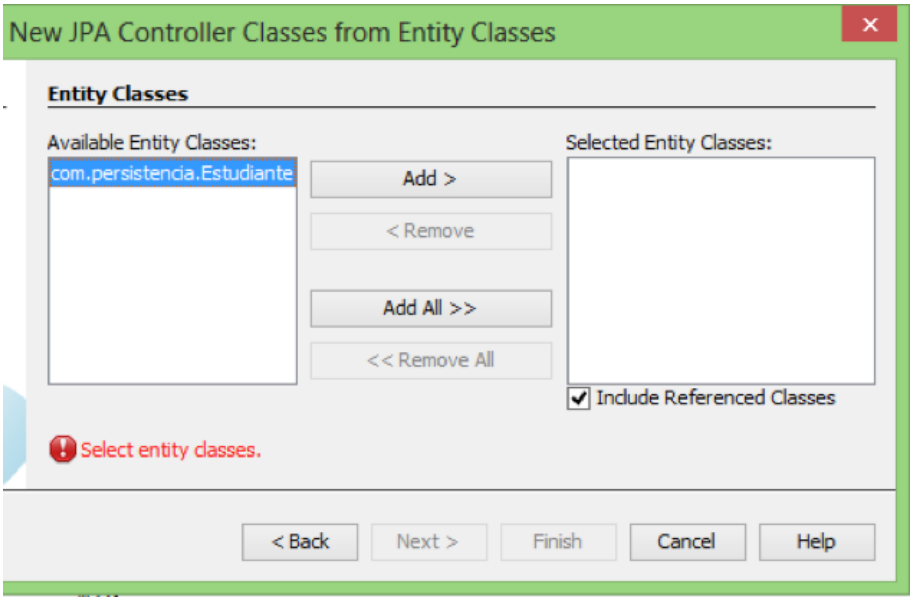


Figura 18. Controlador de la Entity Class, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Verificamos en nombre del proyecto, la ubicación del paquete com.presistencia y damos clic en Finish.

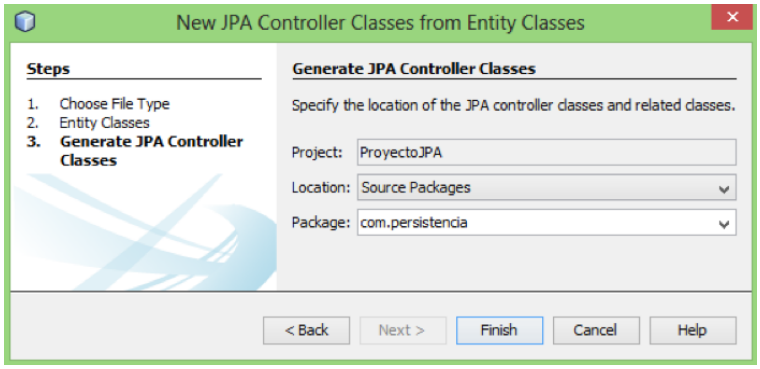


Figura 19. Clase Controlador, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Nos crea el código de la clase controlador EstudianteJpaController.Java, con los métodos necesarios para interactuar con la lógica de persistencia de la entity en la Base de datos, en este archivo encontramos los métodos:

- public void créate: para crear un registro.
- public void edit: para editar los registros.
- public void destroy: para eliminar registros.
- public Estudiante findEstudiante: para buscar registros.
- public List: para listar los registros.

```
public void create(Estudiante estudiante) throws RollbackFailureException, Exception {
    EntityManager em = null;
    try {
        utx.begin();
        em = getEntityManager();
        em.persist(estudiante);
        utx.commit();
    } catch (Exception ex) {
        try {
            utx.rollback();
        } catch (Exception re) {
            throw new RollbackFailureException("An error occurred attempting to roll back the transaction.", re);
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

Figura 20. Método créate, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Este método utiliza una instancia de la clase Estudiante como argumento, invocando al método em.persist(estudiante) que pertenece a la clase getEntityManager.

```
public void edit(Estudiante estudiante) throws NonexistentEntityException, RollbackFailureException, Exception {
    EntityManager em = null;
    try {
        utx.begin();
        em = getEntityManager();
        estudiante = em.merge(estudiante);
        utx.commit();
    } catch (Exception ex) {
        try {
            utx.rollback();
        } catch (Exception re) {
            throw new RollbackFailureException("An error occurred attempting to roll back the transaction.", re);
        }
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Long id = estudiante.getId();
            if (findEstudiante(id) == null) {
                throw new NonexistentEntityException("The estudiante with id " + id + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

Figura 21. Método editar, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Este método utiliza una instancia de la clase Estudiante como argumento, invocando al método em.merge(estudiante) que pertenece a la clase getEntityManager.

```
public void destroy(Long id) throws NonexistentEntityException, RollbackFailureException, Exception {
    EntityManager em = null;
    try {
        utx.begin();
        em = getEntityManager();
        Estudiante estudiante;
        try {
            estudiante = em.getReference(Estudiante.class, id);
            estudiante.getId();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The estudiante with id " + id + " no longer exists.", enfe);
        }
        em.remove(estudiante);
        utx.commit();
    } catch (Exception ex) {
        try {
            utx.rollback();
        } catch (Exception re) {
            throw new RollbackFailureException("An error occurred attempting to roll back the transaction.", re);
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

Figura 22. Método destroy (Long id), código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Este método recibe un parámetro id, que es el campo principal de la entity para recuperar el registro de la Base de Datos.

```
public List<Estudiante> findEstudianteEntities() {
    return findEstudianteEntities(true, -1, -1);
}

public List<Estudiante> findEstudianteEntities(int maxResults, int firstResult) {
    return findEstudianteEntities(false, maxResults, firstResult);
}
private List<Estudiante> findEstudianteEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(Estudiante.class));
        Query q = em.createQuery(cq);
        if (!all) {
            q.setMaxResults(maxResults);
            q.setFirstResult(firstResult);
        }
        return q.getResultList();
    } finally {
        em.close();
    }
}
public Estudiante findEstudiante(Long id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(Estudiante.class, id);
    } finally {
        em.close();
    }
}
```

Figura 23. Métodos de recuperación de registros, código creado por el autor desde el IDE Netbeans
Fuente: Propia.

Creando una aplicación partiendo desde un esquema de base de datos

Esquema de base de datos: es una representación gráfica de la estructura de la Base de Datos.

El esquema de la base de datos puede estar representada de forma gráfica y a través de un Script SQL.

Figura 24. Esquema grafico de la Base de datos, imagen generada por el autor
Fuente: Propia.

Esquema de la Base de Datos Script SQL

```
CREATE TABLE IF NOT EXISTS `estudiante` (
  `IdEstudiante` int(11) NOT NULL,
  `Nombres` varchar(25) NOT NULL,
  `Apellidos` varchar(25) NOT NULL,
  `Direccion` varchar(50) NOT NULL,
  `Telefono` varchar(12) NOT NULL,
  PRIMARY KEY (`IdEstudiante`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- --------------------------------------------------------
CREATE TABLE IF NOT EXISTS `materia` (
  `IdMateria` int(11) NOT NULL DEFAULT '0',
  `Materia` varchar(25) NOT NULL,
  PRIMARY KEY (`IdMateria`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- --------------------------------------------------------
CREATE TABLE IF NOT EXISTS `nota` (
  `IdEstudiante` int(11) NOT NULL DEFAULT '0',
  `IdMateria` int(11) NOT NULL DEFAULT '0',
  `Nota` int(11) NOT NULL,
  PRIMARY KEY (`IdEstudiante`,`IdMateria`),
  KEY `IdMateria` (`IdMateria`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- Filtros para la tabla `nota`
ALTER TABLE `nota`
  ADD CONSTRAINT `nota_ibfk_2` FOREIGN KEY (`IdMateria`) REFERENCES `materia` (`IdMateria`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `nota_ibfk_1` FOREIGN KEY (`IdEstudiante`) REFERENCES `estudiante` (`IdEstudiante`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Figura 25. Esquema de la Base de Datos en Script SQL, imagen generada por el autor
Fuente: Propia.

En Netbeans crearemos las entities JPA de modo automático a partir del esquema y las aplicaciones Java Server Faces de las entidades JPA.

Abrimos el archivo universidad.sql, damos clic en File, Open Archivo y seleccionamos universidad.sql, para crear la BD vamos la pestaña Services y seleccionamos Java DB con el botón derecho del mouse, y damos clic en créate Database ….

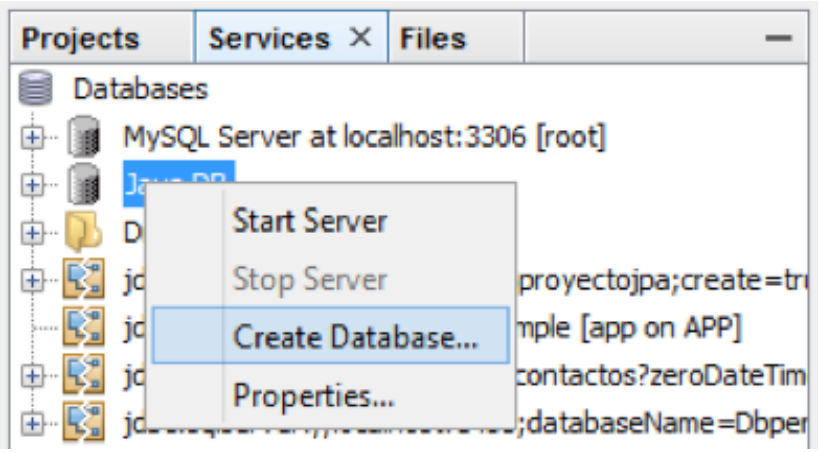


Figura 26. Creando la Base de Datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Creamos la base de datos con los siguientes campos:

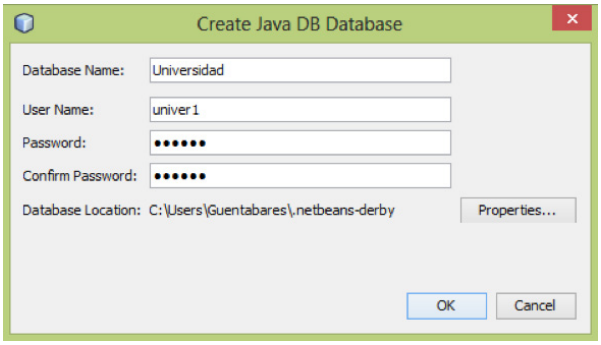


Figura 27. Parámetros de la Base de Datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Base de datos: Universidad
User Name: univer1
Password: 123456

Ya tenemos la BD para ejecutar el esquema SQL para crear las tablas, para la creación de las tablas desplegamos en jdbc:derby://localhost:1527/Universidad, donde está la base de datos, damos clic en tables con el botón derecho del mouse y seleccionamos créate table….

Adicionamos cada table de nuestra base datos para nuestro ejercicio vamos a crear las tablas estudiantes, materia, nota, como muestra la figura.

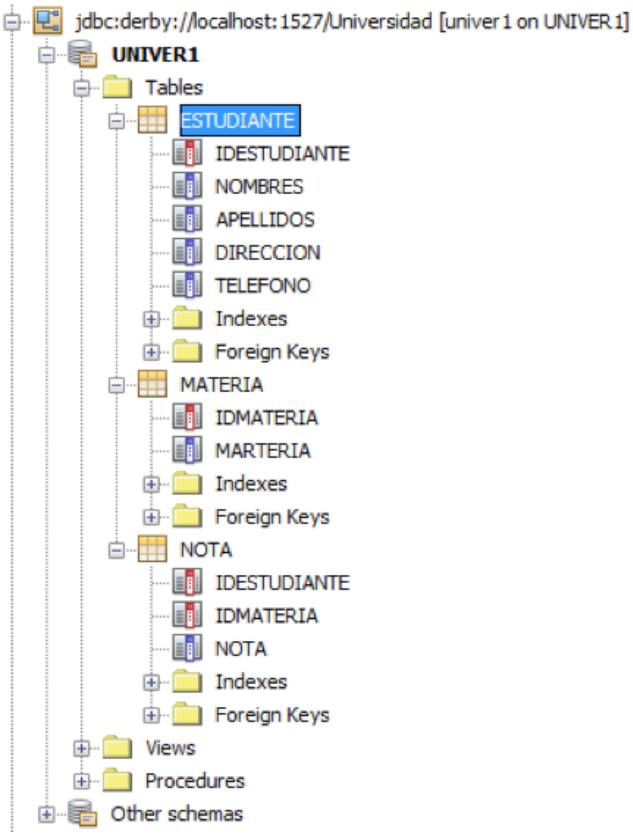


Figura 28. Estructura de la Base de Datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Persistencia en nivel web

Para crear las entities de la base de datos vamos a crear un proyecto web application, para esto creamos nuevo proyecto.

Inicialmente se debe crear un proyecto de aplicación Web, con el nombre "UniversidadJPA".

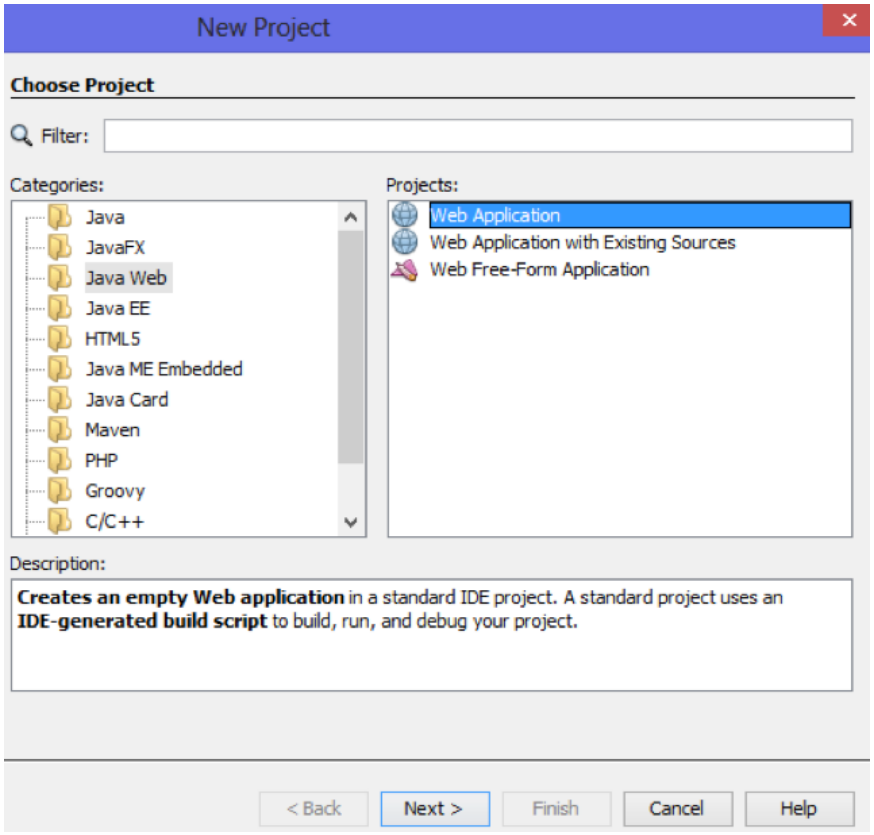


Figura 29. Creación de una aplicación web, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Clic en Next> y seleccionamos como frameworks JavaServer Faces

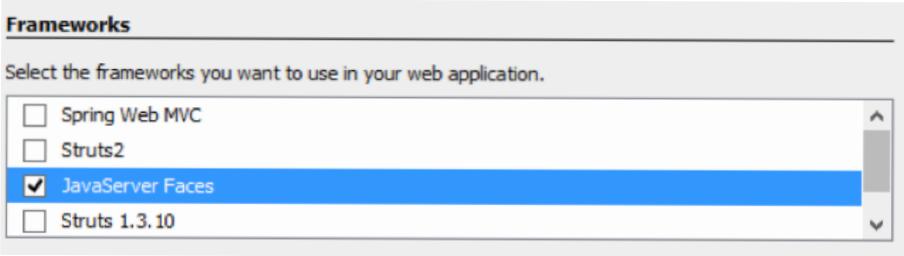


Figura 30. Frameworks JavaServer Faces, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Dejamos por defecto las demás opciones y damos clic en Finish.

Para crear una entity JPA creamos un nuevo archivo y buscamos dentro de la categoría la opción de "Persistencia" y tipo de archivo "Entity Class".

Clic en botón Next>, asignamos nombre a la clase "Estudiante", y creamos un paquete llamado com.persistencia, verificamos que este seleccionado la opción "Create Persistence Unit", parra cada entity class se crea una unidad de persistencia bajo un archivo xmly damos clic en Next>.

Crear las entities de la base de datos creamos un archivo nuevo, clic en file –> New File, seleccionamos la categoria "Persistence" y en tipos de archivos "Entity Classes from Database". Clic en Next>.

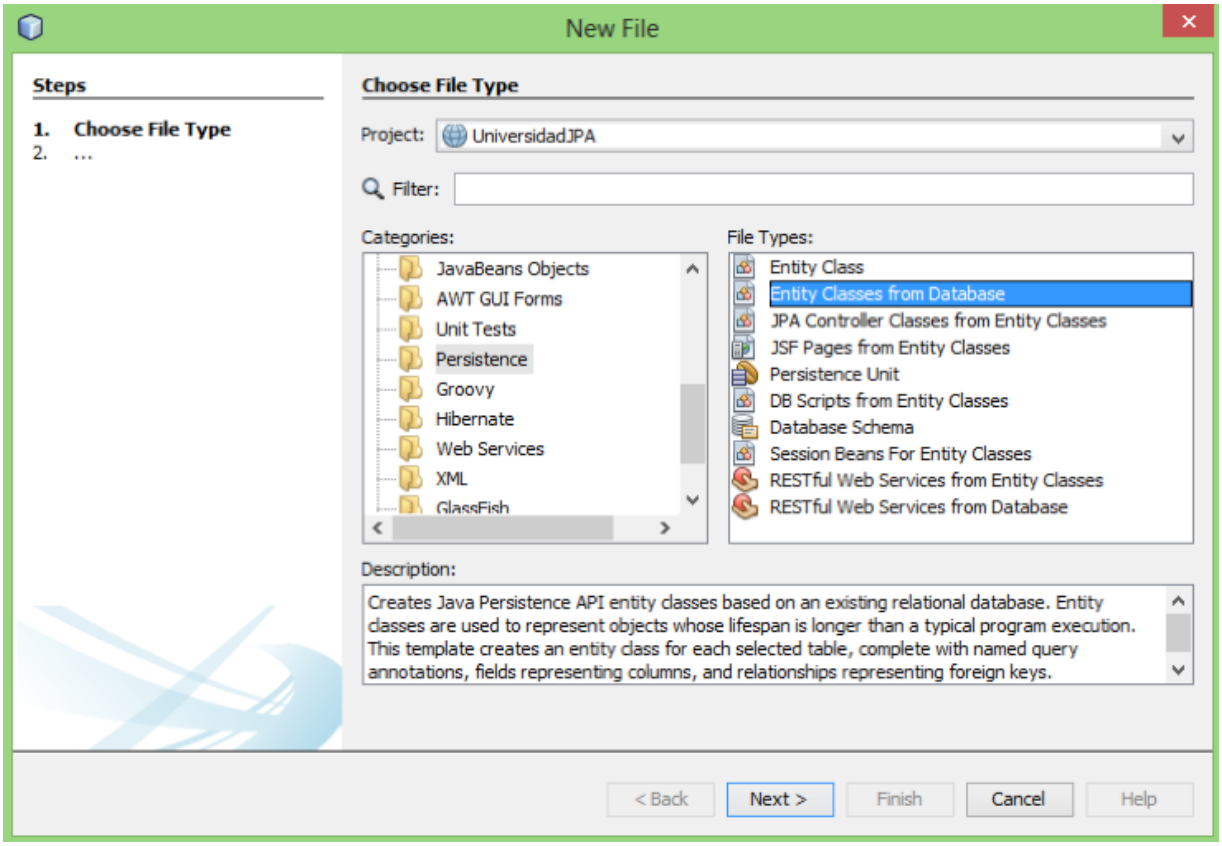


Figura 31. Creando archivo entity Classes from Database, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Escogemos la opción de "New Data Sourde" de la ventana Data Source para asignar la fuente de la base de datos.

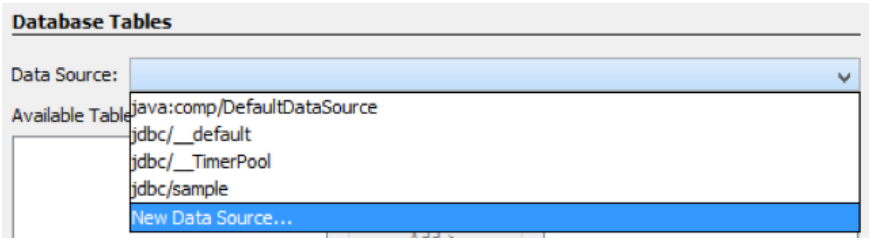


Figura 32. Nueva fuente de datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Asignamos un nombre al JNDI Name: y seleccionamos la conexión a la Base de Datos.

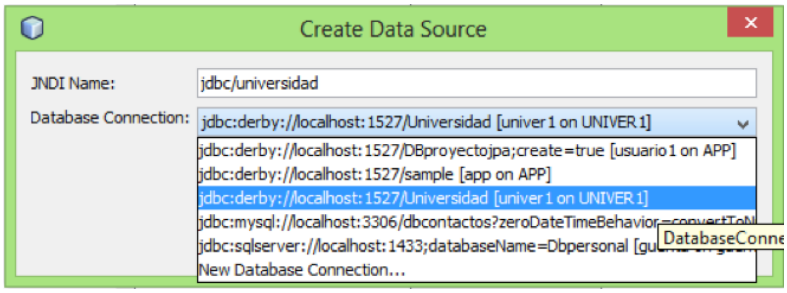


Figura 33. Selección del jdbc:derby, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Nos carga la tablas que integran la base de datos, las cuales seleccionamos y add all >>. Y clic en Next>>.

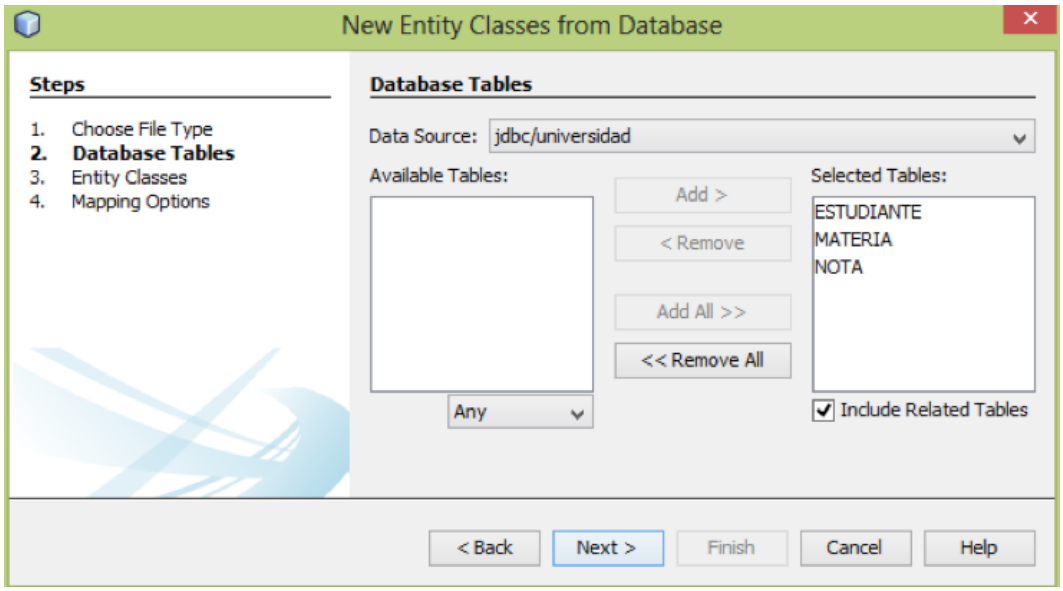


Figura 34. Selección tablas de la base de datos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

En la ventana siguiente nos permite cambiar el nombre de las clases de cada tabla de la base datos, las dejamos por defecto, creamos un paquete para las clases.

"com.jpa". clic en Next>> y Finish.

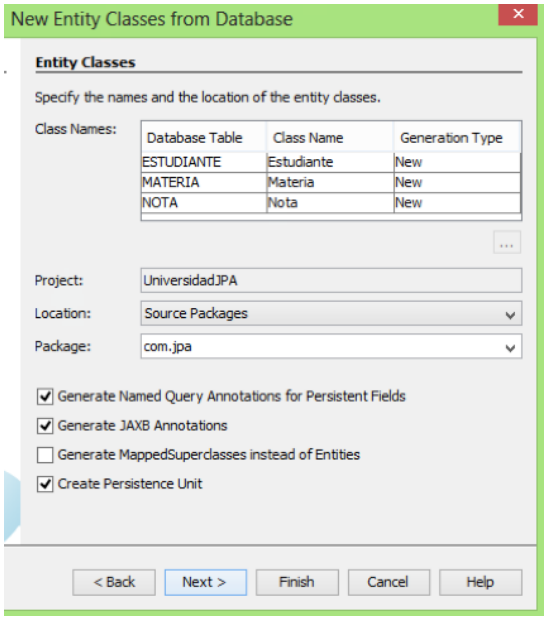


Figura 35. Clases Entity, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Podemos observar en el proyecto la creación de las entities para cada una de las tablas que conforma la base de datos.

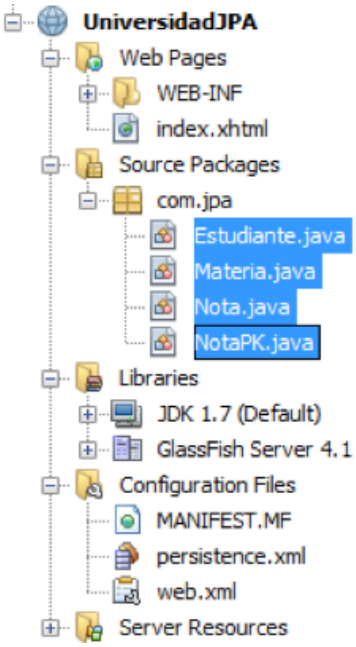


Figura 36. Proyecto con lista de clases de las entidades de la tabla, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ahora crearemos las aplicaciones JSF para cada una de las entidades JPA.

Para esto creamos un nuevo archivo file -> New file, seleccionamos la categoria JavaServer Faces y en tipos de archivos seleccionamos JSF Pages from Entity Classes.

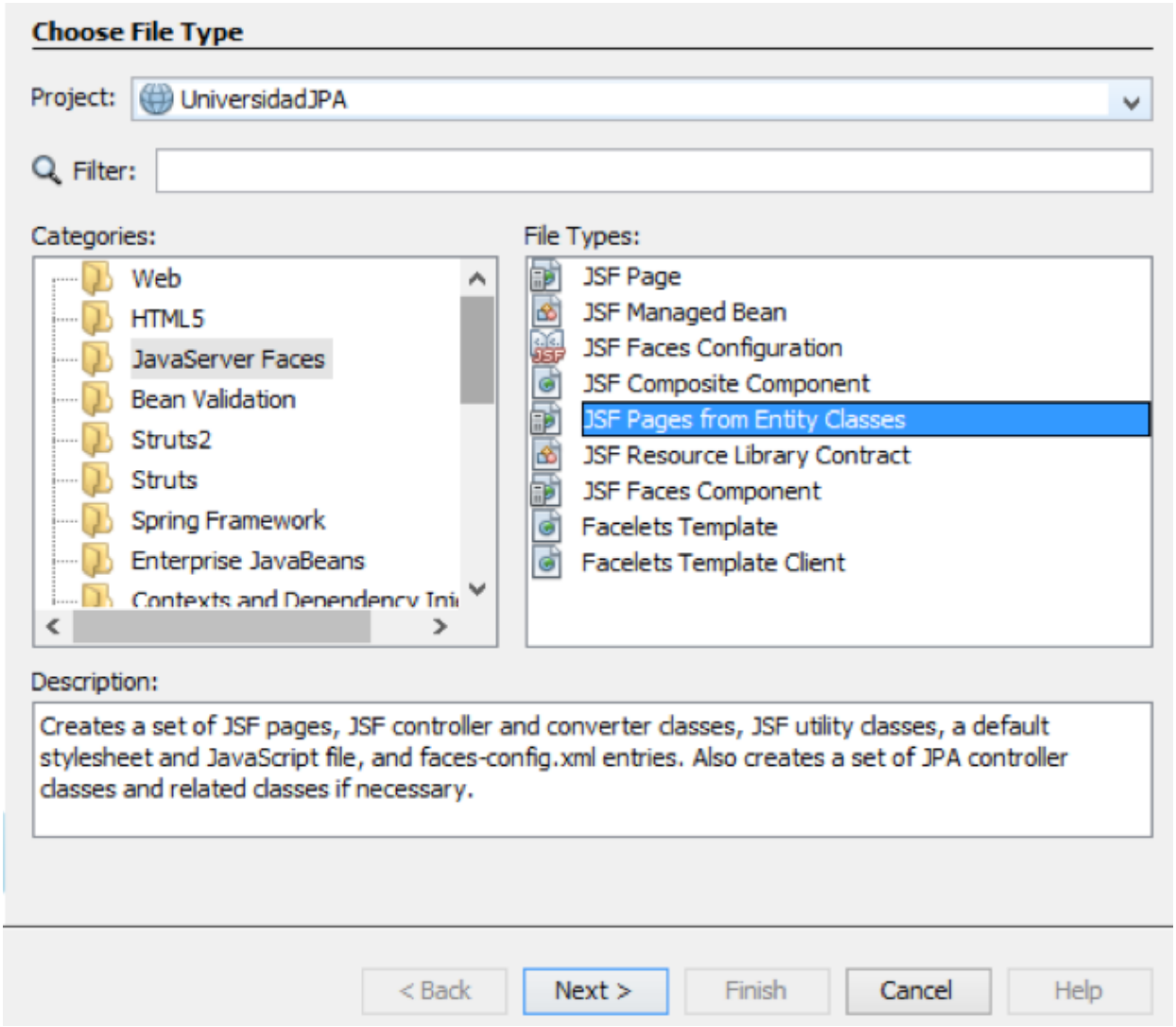


Figura 37. Creación de las Pages from Entity Classes, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

En la siguiente ventana nos permite seleccionar las entities para la creación de las páginas.

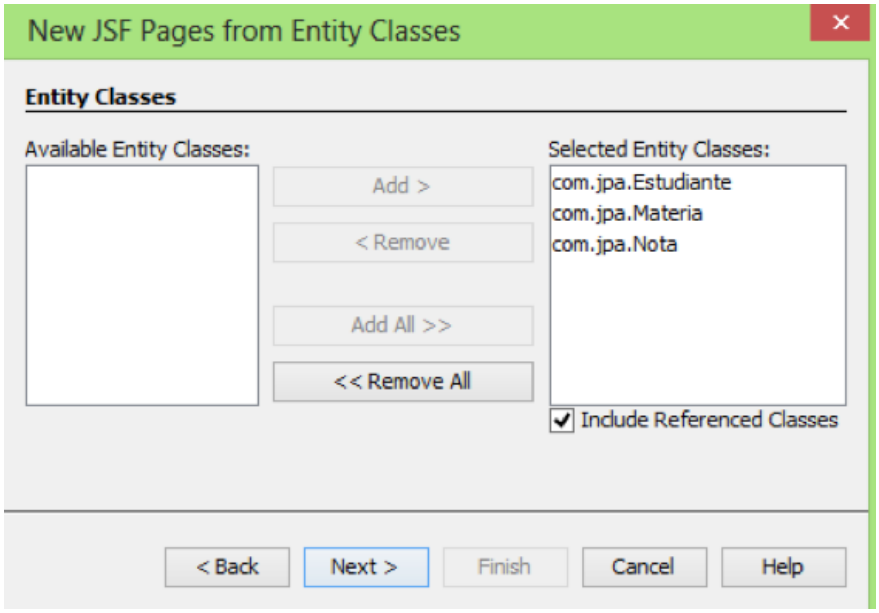


Figura 38. Selección de las entities JPA, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Verificamos el nombre del proyecto y podemos cambiar los paquetes para los Session Bean Package y las Classes Package de la aplicación, las Sessions Bean es el encargado de la controlar la lógico del modelo con la base de datos. Y clic en Finish.

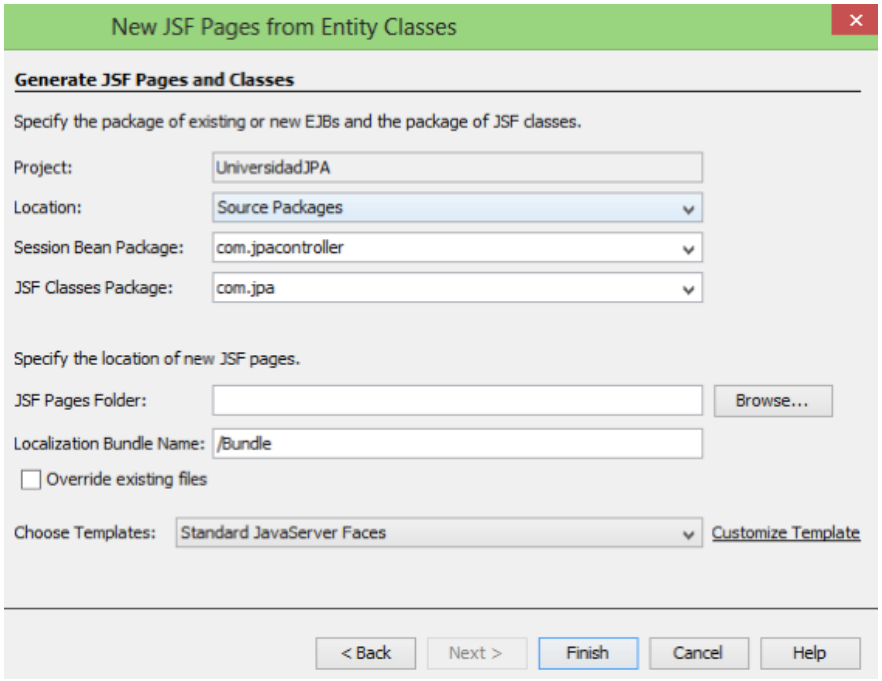


Figura 39. Definición del package de la Sesión Bean y las Classes Package, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ya tenemos nuestra aplicación con los archivos generados de cada una las entities JPA, los controladores, las páginas web para cada entidad (Create.xhtml, Edit.xhtml, List.xhtml, View.xhtml). Aplicación CRUD, la cual podemos ejecutar para ver su funcionamiento.

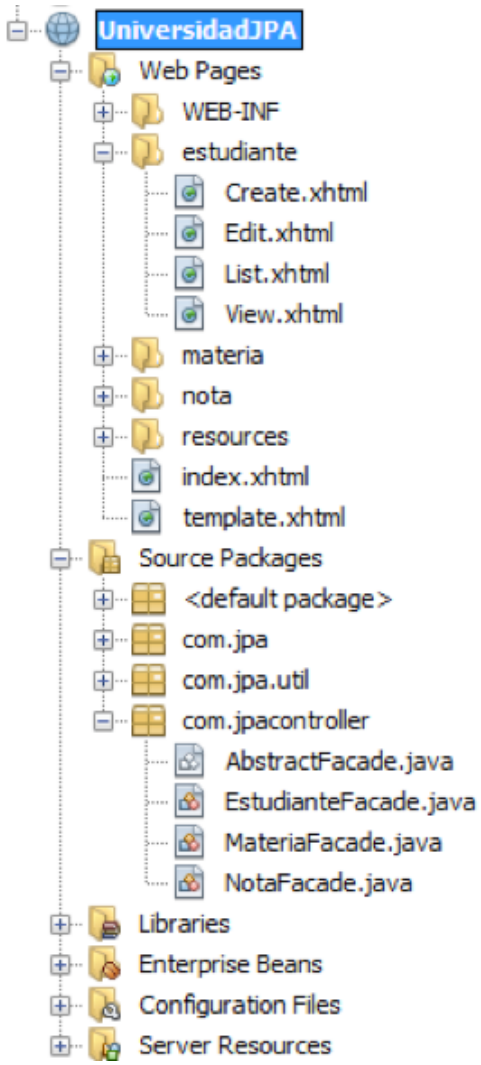
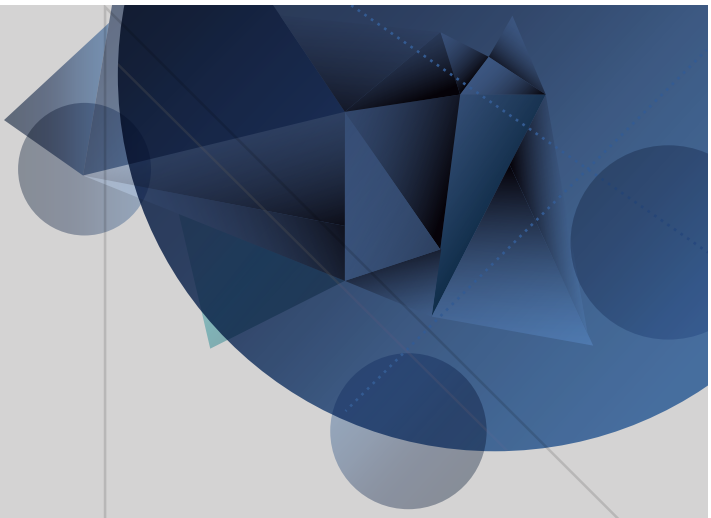


Figura 40. Estructura de archivos de la aplicación UnivesidadJPA, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

4

Unidad 4

Seguridad web



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El uso de la internet, ha abierto multiples alternativas para los usuarios, desde el comercio, tranferencias de datos, procesamiento de informacíon, etc. El desarrollo de aplicaciones cada vez debe garantizar a los usuarios la confiablidad y el buen uso de las aplicaciones web. Por eso es importante la seguridad que debe rodear y proteger estas aplicaciones y no permitir que usuarios mal intencionados hagan uso idenbido de la información y que esta quede alamcenada correctamente en los repositorios de bases de datos de forma segura.

Durante este curso el estudiante comprendera la importancia de la seguridad en aplicaciones web y la vulnerabilidad que pueden llegar a presentar al ser accedidas atraves de las redes.

U4 Metodología

Se recomienda seguir de forma ordenada la lectura de la unidad, y analizar de forma cuidadosa los ejemplos de códigos donde se plasma tema, logrando comprender el concepto y permitiéndole avanzar de forma clara durante el desarrollo de la unidad. Es importante apoyar el tema con lecturas recomendadas en la bibliografía y en los recursos de las lecturas complementarias.

U4 Desarrollo temático

Seguridad web

Una aplicación web segura puede ser tan compleja como necesite el sistema, requiere de conocer los puntos vulnerables de la seguridad. Identificar de forma clara, las posibilidades de seguridad que ofrecen algunos sistemas operativos y aplicarlas funciones de seguridad para contrarrestar las amenazas.

En la siguiente lista encotraremos algunas pautas que seguridad minima que deben aplicarse a una aplicación web y que se deberian de tener en cuenta para dar mayor seguridad y las cuales deberiamos de seguir.

- Recomendaciones generales de seguridad para aplicaciones web.
- Ejecutar aplicaciones con privilegios mínimos.
- Conocer a los usuarios.
- Protegerse contra entradas malintencionadas.
- Tener acceso seguro a bases de datos.
- Crear mensajes de error seguros.
- Mantener segura la información confidencial.
- Usar cookies de forma segura.
- Protegerse contra amenazas de denegación de servicio (Microsoft, 2007).

La arquitectura de las aplicaciones web típicas que utiliza una base de datos tiene la siguiente estructura:

Encontramos tres niveles

1. Interfaz de Usuario: la encargada de gestionar la interacción entre el cliente y el sistema, en una aplicación web es el navegador de internet.

Imagen 1. Interfaz de usuario, Imagen tomada del sitio http://www.staffcreativa.pe/blog/disenar-interfaz-usuario/

2. Lógica de la aplicación: se ejecuta en los servidores web y servidores de aplicaciones, con el uso de diferentes tecnologías podrán generar conocimiento y procesar información con un fin concreto.

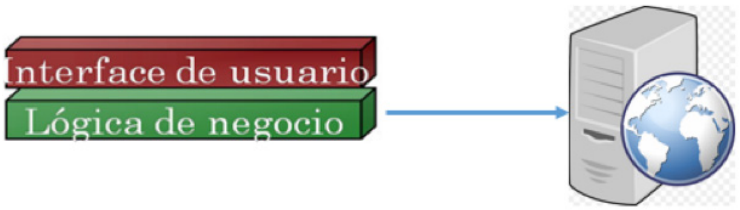


Imagen 2. Nivel lógica de la aplicación
Fuente: Propia.

3. Almacenamiento de datos: es el repositorio de la información donde se guarda la información de una organización. Que puede estar bajo diferentes tecnologías de almacenamiento, como son: LDAP, Base de Datos Relacional, XML o un fichero de datos.

Imagen 3 Nivel del almacenamiento
Fuente: Propia.

Existe una conexión entre cada uno de los niveles adyacentes . el usuario a traves de la información que envia por medio del navegador de internet interactua con el servidor web y a su vez la lógica de la aplicación interactua con el almacenamiento de datos para consultar y almacenar la información en el sistema. Cada iteración tiene integrada un conjunto de protocolos y lenguajes entre el nivel de interface y el nivel de aplicación encontramos los protocolos HTTP Y HTTPS, para enviar la información. El servidor web y los repositorios de datos se lanzan consultas en lenguajes propios sobre protocolos de red creados especificamente para esta función.

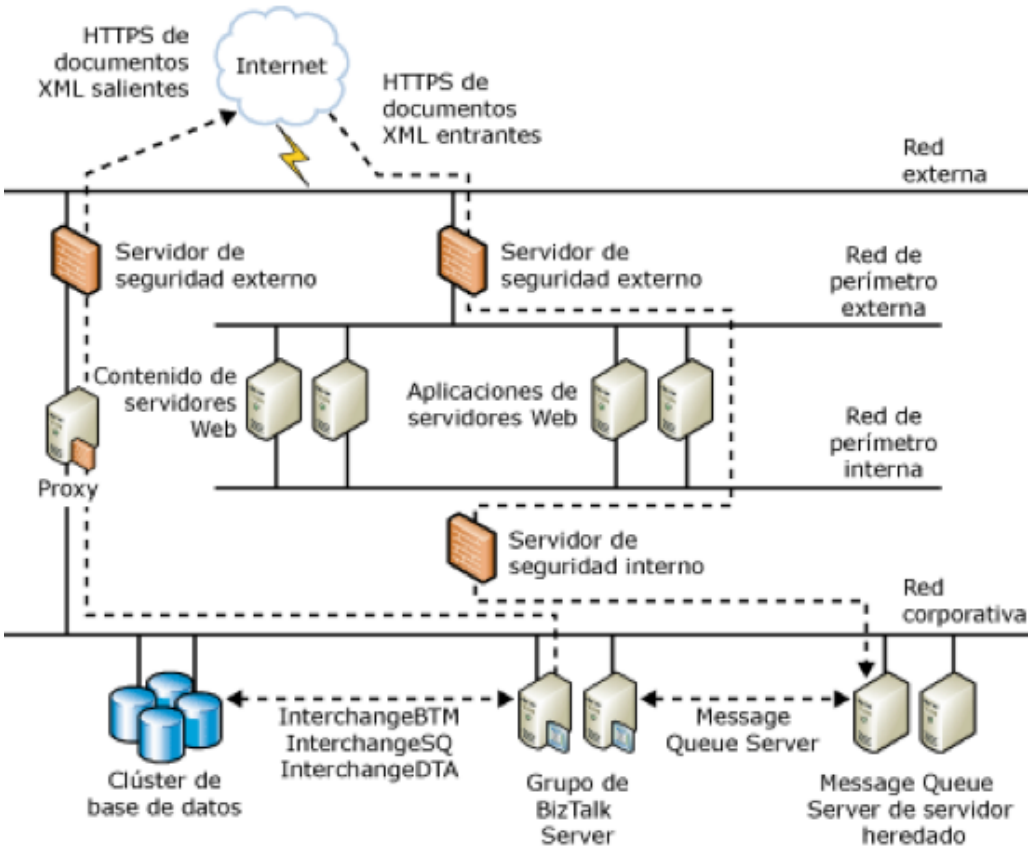


Imagen 4. Arquitectura de seguridad
Fuente: https://msdn.microsoft.com/es-es/library/aa577397.aspx

La gráfica muestra la arquitectura de seguridad, ha sido segmentado y controlado por firewalls, protegiendo los servidores de contenido, la base de datos etc.

En una base de datos relacional la lógica de la aplicación lanzara consultas en lenguaje SQL sobre protocolos como Tabular Data Stream Oracle Net y en el caso de un repositorio en formato por XML utilizará consultas XPATH.

Cuando se presentan fallas en una aplicación web, permitiendo el acceso indebido de usuarios no autorizado para estraer información de un sistema informático, se debe en una alta probabilidad de un fallo de seguridad en el codigo de la aplicación.

Los casos más comunes de ataques a las aplicaciones web según la OWASP (Open Web Application Security Project) son:

- Injection.
- Cross-site Scripting (XSS).
- Broken Authentication and Session Management.
- Insecure Direct Object References.
- Cross-Site Request Forgery (CSRF).
- Security Misconfiguration.
- Inseccure Cryptographic Storage.
- Failure to Restrict URL Access.
- Insufficient Transport layer Protection.
- Unvalidated Redirects and Forwards (Institute, 2012).

A traves de estos ataques, intentan explotar fallos de seguridad de aplicaciones web en tecnologías de cliente o servidor, con la intención de extraer informacion de la base de datos.

Inyeccion de código SQL en aplicación web

Se presenta cuando una aplicación web no validad correctamente los datos que introduce el usuario en la interfaz, y que se utilizan en consultas a repositorios de datos.

Imagen 5. Formulario Login de aplicación Web
Fuente: Propia.

Un sistema web, que tiene un sistema de validación de usuario, a traves de un formulario donde se digita el login y un password, sino se valida por parte del programador correctamente la información que recibe del formulario a traves de una concatenación, construyendo la consulta que se envia al servidor de la Base de Datos, el atacante puede interactuar directamente con la Base de Datos, como por ejemplo inyectando consultas SQL extrayendo, modificando y eliminando información.

Imagen 6. Inyeccion de SQL en el password
Fuente: Propia.

Los ataques a traves de estos formularios, se dan cuando el atacante no introduce una contraseña correcta, al contrario digito una condición utilizando un carácter especial de comillas simple.

Seguridad en aplicaciones Java

Nos centraremos en el tema relacionado con la seguridad que maneja la plataforma Java, veremos las caracterisiticas que hacen de este un lenguaje de programación seguro.

- La Máquina Virtual de Java como entorno de ejecución seguro, la ejecución de la JVM en el sistema, no es un sistema interpretado ni compilado, funciona como un sistema hibrido, el desarrollo de las aplicaciones en Java se compilan totalmente independiente de la máquna en un proceso denominado bytecode, y estos son interpretados de forma simple por la Máquina Virtual, que se encuentra activo en la plataforma que trabajamos. La responsabilidad de la seguridad que tiene incorporado Java esta en la Máquina Virtual, por lo que es necesario su correcta implementación.

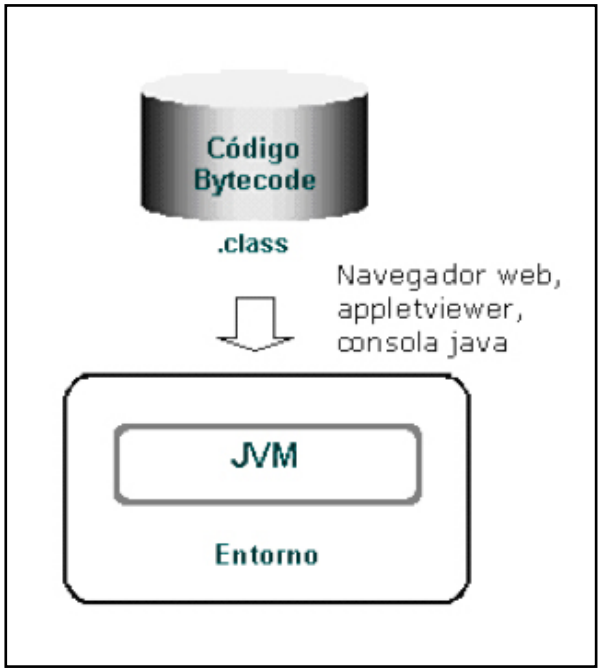


Imagen 7. Máquina virtual de Java
Fuente: http://Java.ciberaula.com/articulo/tecnologia_Java/

- Conjunto de interfaces y arquitecturas que son proporcioandas con el programador de aplicaciones Java API, El API es un biblioteca de componentes que se agrupan en liberias o paquetes que se integran en las aplicaciones, proporcionando una gran variedad de utilidades para la construcción de interfaces gráficas, acceso a bases de datos, redes, entrda y salidad E/S, etc. La API tambien nos proporcionan librearias o paquetes que implementan o permiten el acceso a elementos de seguridad en varios niveles, en los que podemos mencionar los algoritmos de encriptación, certificados y firmas digitales.

La segurida en Java la podemos analizar desde varios frentes:

Desde el entorno de ejecución y desde las interfaces y arquitectura de seguridad

Enterno de ejecucuión: inicialmente la seguridad para los usuarios esta en la máquina virtual, como mediadora y controladora de lo que se puede ejecutar y bajo que cirscunstancias se ejecuta.

Interfaces y arquitectura de seguridad: la implentación de técnicas criptograficas permiten garantizar la seguridad en aplicaciones cliente-servidor. Esta tecnica las proporciona Java desde su arquitectura para integrar en sus aplicacines componentes seguros que den confialbidad al uso de la aplicación.

Una máquina virtual de Java contiene un mecanismo de verificador de clases, el cual permite asegurar, al cargar las clases, si estas tienen o cumplen con la estructura interna correcta. Al detecrar un patron o un problema dentro de la clase se genera un excepción, ya que la estructura de la clases se genera con una estructurra de bytes. La máquina virtual no puede determinar sin una clase en realidad es un bytecode, con relación a esto la máquina virtual tiene un verificador de clases que se activa sobre clases no seguras, permitiendo la corrección del código.

Uno de los objetivos del verificador de la máquina virtual de Java es garantizar la construcción de aplicaciones robustas, bien estructuradas y seguras en su codificación, cuando el verificador detecta un codigo mal intecionado en algun método, generando una incosistencia en la ejecuación de la clase la máquina virtual no funcionaria si este es invocado.

La verificación del bytecode generado se realiza despues de ser cargado, donde el verificar inicia su tarea de comprobación en dos fases.

1. Se realiza despues de cargar la clase, comprueba la estructura interna de la clase y la integrigirdad del bytecodes.
2. Se llave a cado durante la ejecución del bytecode generado de la clase, confirmando las referencias simbolicas de las clases, atributos y métodos.

El cargador de clases en Java

Es el que se encarga de cargar los bytecodes definidos que definen la clase, para ser verificados y poder generar las clases finales. La máquina virtual incluye un cargador de clase

primario que a su vez es la clase en Java verificadora de el bytecode de las clases creadas de la aplicación y el cargador de clase primario esta desarrollado en C.

La seguridad tambien esta presente en el cargador el cual cumple con las siguientes tareas:

- El cargador primario es el encargado de invocar incialmente el paquete Java.*, donde todas las clases tienen un apuntador null. Para la seguridad es importante debidio a que son las clases basicas de inicialización y correcto funcionmianto del sistema y son cargadas desde el sistema de archivos local evitando que sean cargadas remotamente o reemplazadas.
- El cargador facilta espacios de nombre de clases diferentes para ser cargadas, evitando la duplicidad, las coliciones entre los nombre de clases cargas desde difretnes origenes.
- Dentro de la máquna virtual no pueden cargase ni comunicarse clases provenientes de fuentes diferentes, evientando la obtención de información de aplicaciones no fiables.

ClassLoader

Es una clase que carga otras clases, ClassLoader se encarga de verificar las clases de una aplicación en memoria según el requerimiento que tenga la máquina virtual Java.

Cuando se lanza un programa por consola se ejecuta la siguiente estructura.

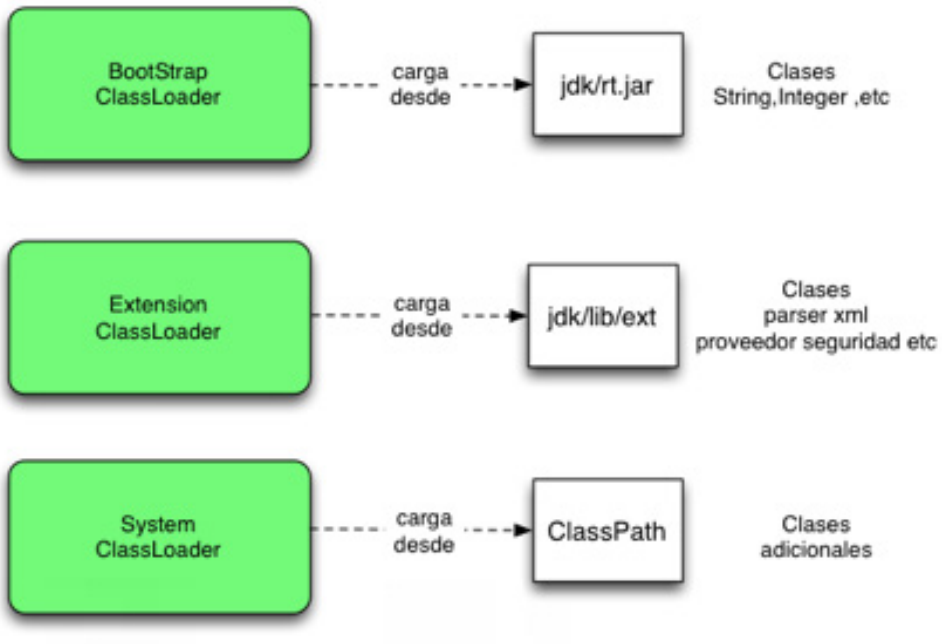


Imagen 8. Estructura de ejecución de una clase Java
Fuente: http://www.arquitecturaJava.com/el-concepto-de-classloader/

Una de la cosas más importantes entre objetos son las relaciones que se dan en la ClassLoaders, entre si como padre/hijo.

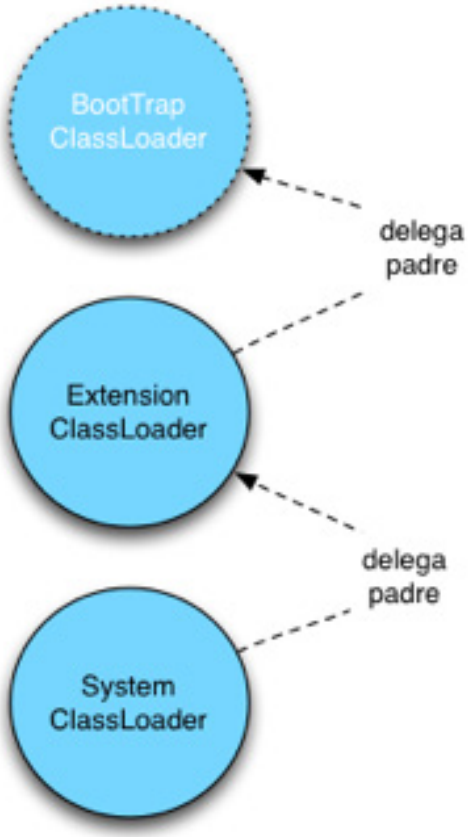


Imagen 9. Relacion Padre/Hijo
Fuente: Propia.

Cuando una clase Loader va a cargar una clase, esta consulta a su claseLoader padre y si la tiene disponible la carga. En caso de no poderla cargar este consulta a la siguiente claseLoader hasta llager a BootTrap-ClassLoader.Si ninguno de los ClassLoader padre puede cargar la clase, el cargador de clases actual intentará cargarla de sus rutas disponibles. Sino la encuentra se procudirá un ClassNotFoundException (Caules, 2013).

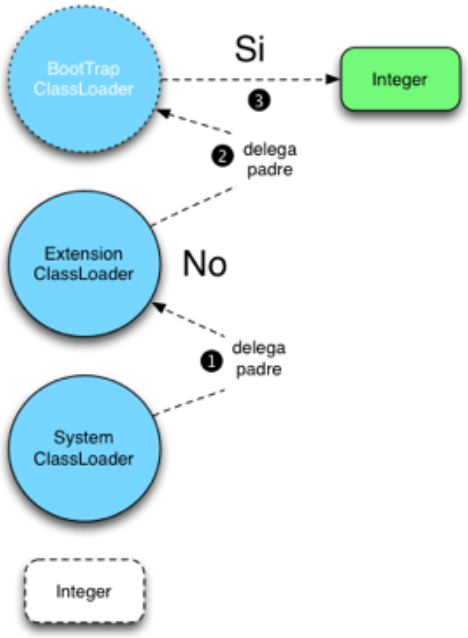


Imagen 10. Delegación de la classLoader
Fuente: http://www.arquitecturaJava.com/el-concepto-de-classloader/

Cuando se carga la clase persona por la ClassLoader:

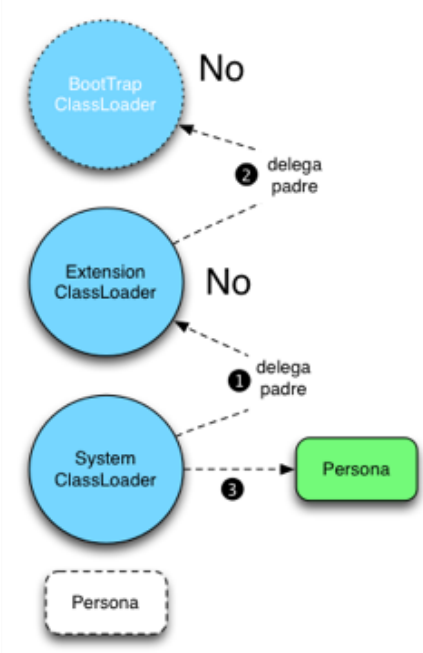
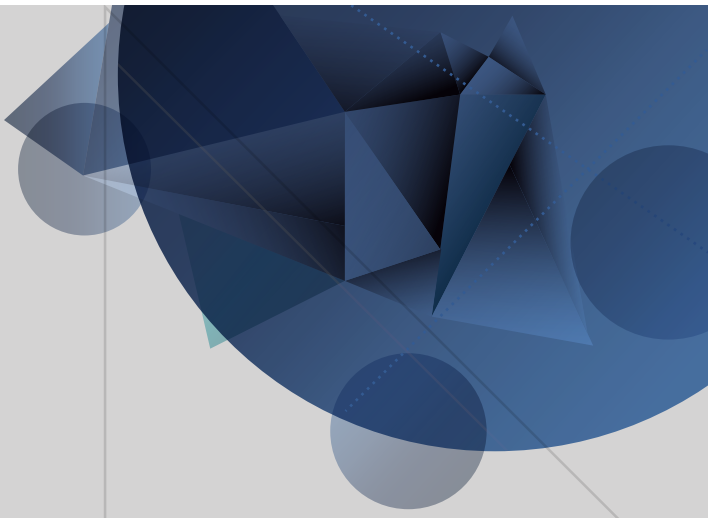


Imagen 11. Cargar la clase Persona
Fuente: http://www.arquitecturaJava.com/el-concepto-de-classloader/

4

Unidad 4

Seguridad en aplicaciones web



Modelos de programación II

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La protección de las aplicaciones web es un requerimiento que debe estar muy presente entre el desarrollo de dichas aplicaciones, los constantes ataques que sufren las aplicaciones por diferentes medios, es sin lugar a dudas el quehacer de los atacantes a diario.

Es por esto que aplicaciones deben estar desarrolladas con los niveles de calidad cumpliendo los estándares en el desarrollo de software, que den confiabilidad al usar la aplicación por parte del usuario.

En este curso vamos a tratar el tema de protección de aplicaciones web, con el objetivo de desarrollar aplicaciones más seguras.

U4 Metodología

Es importante llevar a cabo el desarrollo de la semana de una forma secuencial, para tener una claridad en el desarrollo del tema, se recomienda no saltar entre páginas, pues el desarrollo del tema está enfocado al seguimiento paso a paso de una actividad.

Es importante apoyar el tema con lecturas recomendadas en la bibliografía y en los recursos de las lecturas complementarias.

Seguridad en aplicaciones web

La protección de las aplicaciones web, es un requerimiento muy común consiste en restringir el acceso a las páginas para ciertos usuarios. En Java para lograrlo debemos de crear un dominio de seguridad en el servidor donde va a tener la aplicación.

Los dominios de seguridad hacen referencia a colecciones de usuarios y grupos de seguridad, cada dominio de seguridad permite a la aplicación obtener información de seguridad de algún tipo y almacenarla permanentemente.

Esta información de seguridad puede ser almacenada desde distintos dispositivos, desde un archivo plano a una Base de Datos Relacional o un Repositorio. El configurar la aplicación para que obtenga la información segura de un almacenamiento persistente nos libera de mucho trabajo, solo debemos definir la autenticación de la aplicación para que use un dominio de seguridad.

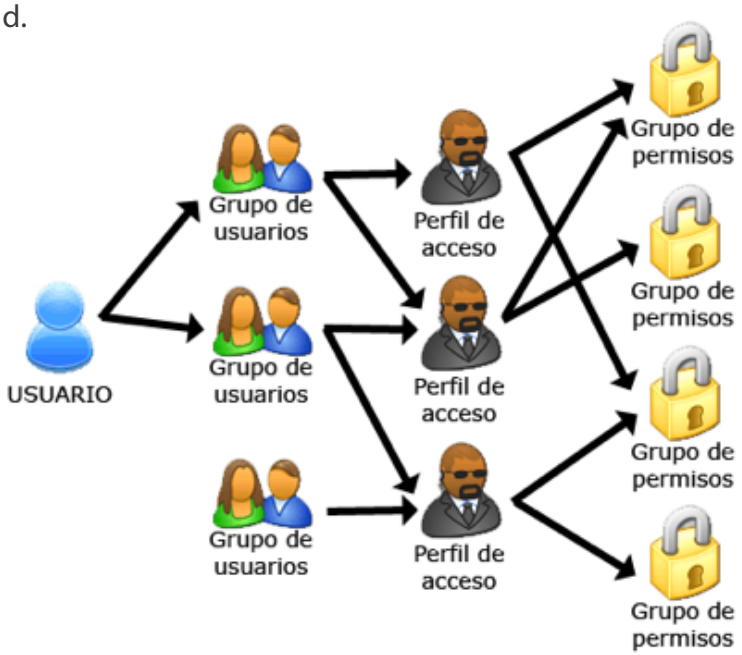


Imagen 1. Grupos de usuarios
Fuente: http://www.metaspaceportal.com/metaspace/portal/15043/15583-users-and-roles?pms=1,15138,15060002,view,normal,0

Cada usuario puede pertenecer a uno o a más grupos de seguridad, las páginas web solo van a ser accesibles para ciertos grupos de seguridad, estos procedimientos de configuración para los dominios de seguridad varían dependiendo del servidor que se esté utilizando.

El Servidor GlassFish tiene un dominio de seguridad ya configurado llamado "file", podemos ingresar también a la consola de administrador del servidor para personalizar los usuarios que tendrán acceso a la aplicación.

Existen diferentes modos para autentificar a los usuarios, el más común es el autenticación basada en formularios, lo que requiere de una página HTML o JSP, para capturar los datos de autenticación que digite el usuario, además de capturar la información de proporcione el usuario se puede encriptar la información configurando las páginas para que usen HTTP/ HTTPS sobre SSL.

En el siguiente diagrama se detalla el proceso que se llevara desde un cliente cuando se loguea en una aplicación.

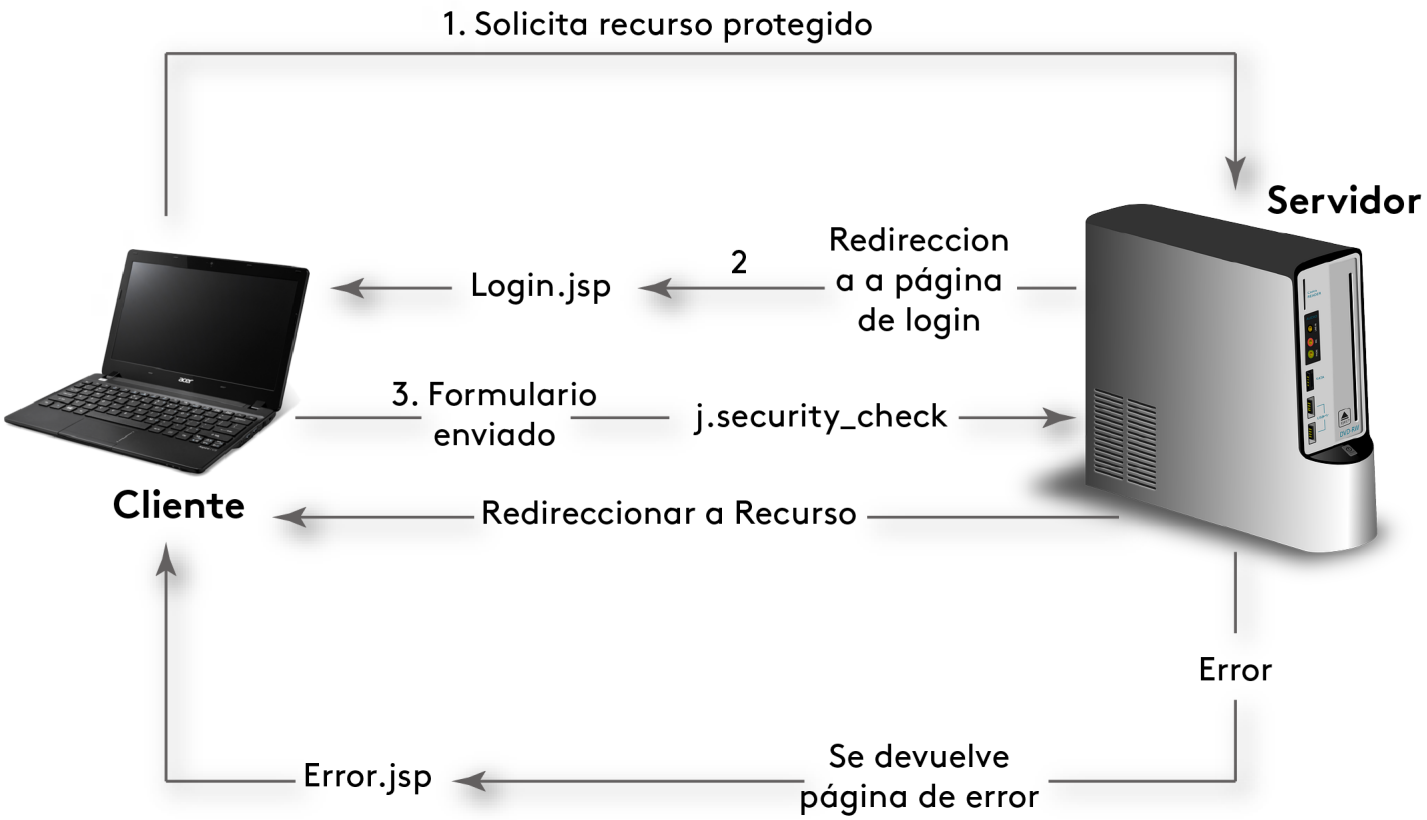


Imagen 2. Proceso de logueo con j.security_check
Fuente: https://rekkeb.wordpress.com/2009/04/18/j_security_check-conocer-la-pagina-original-solicitada/

Los pasos para llevar a cabo este proceso, se debe de crear inicialmente una página de login y otra página para el manejo de errores, cuando no se dé un login correcto. La aplicación web debe ser configurada en el servidor para que pueda usar un dominio de seguridad y el que se encargara de la autenticación.

El primer paso es iniciar el IDE de desarrollo NetBeans y creamos un nuevo proyecto, Web Application, al cual llamaremos securitylogin, clic en Next>

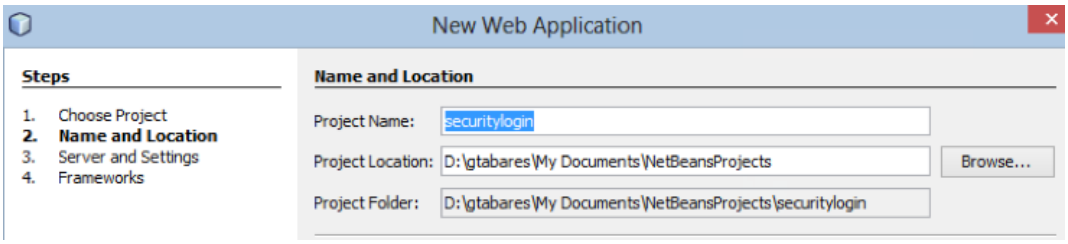


Imagen 3. Nuevo proyecto Web Application, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Seleccionamos el servidor GlassFish y clic en el botón Finish.

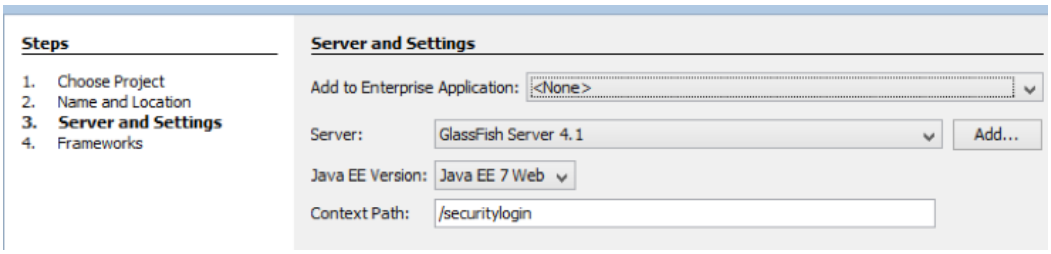


Imagen 4. Servidor GlassFish, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Creamos dentro del proyecto una archivo jsp llamo login.

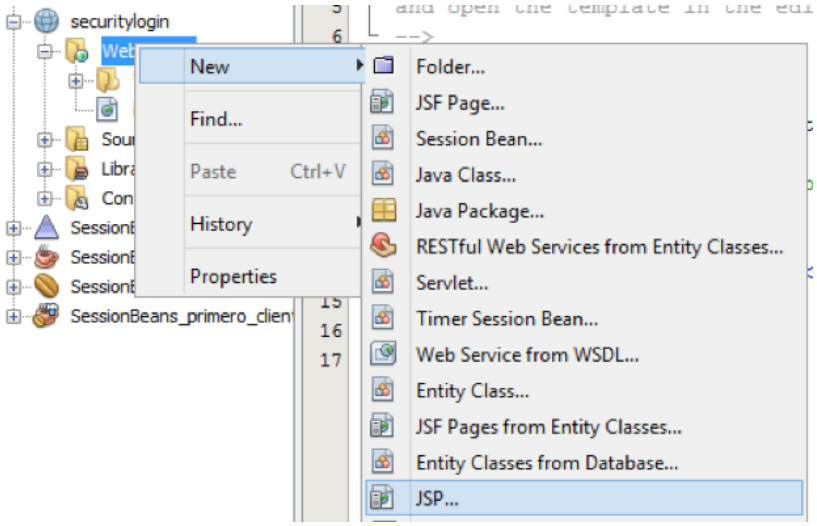


Imagen 5. Creación del archivo jsp, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Asignamos el nombre login.jsp y damos clic en Finish.

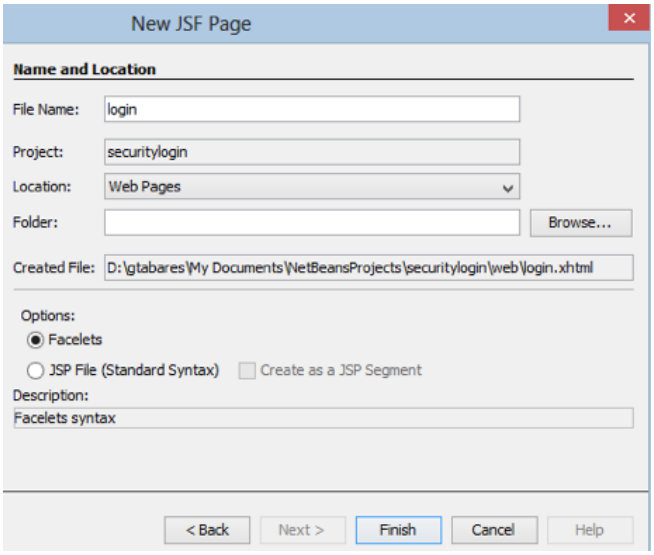


Imagen 6. Creando el archivo login.jsp, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Modificamos el código del archivo login.jsp que nos permita crear un formulario para ingresar datos del login, como muestra la siguiente figura.

```
<%@page language="java" contentType="text/html; charset=utf-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Login</title>
    </head>
    <body>
        <p>Escribe el nombre de usuario y contraseña para ingresar </p>
        <form method="POST" action="j_security_check">
            <table cellpadding="0" cellspacing="0" border="0">
                <tr>
                    <td aling="right">Nombre Usuario: </td>
                    <td><input type="text" name="j_username"></td>
                </tr>
                <tr><td align="right">Contraseña: </td>
                    <td><input type="password" name="j_password"></td>
                </tr>
                <tr><td></td>
                    <td><input type="submit" value="Login"></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Imagen 7. Código formulario login.jsp, código creado por el autor desde el IDE Netbeans
Fuente: Propia

La acción del formulario action="j_security_check" es compactible con los desarrollos de aplicación del servidor en Java Entrerprise Edition - JEE, este cuenta con un servlet de seguridad implementado en la instalación, este servlet es mapeado a la URL j_security_check.

En el siguiente paso se crea la página de loginerror, damos clic derecho y creamos un archivo jsp en nuestra aplicación.

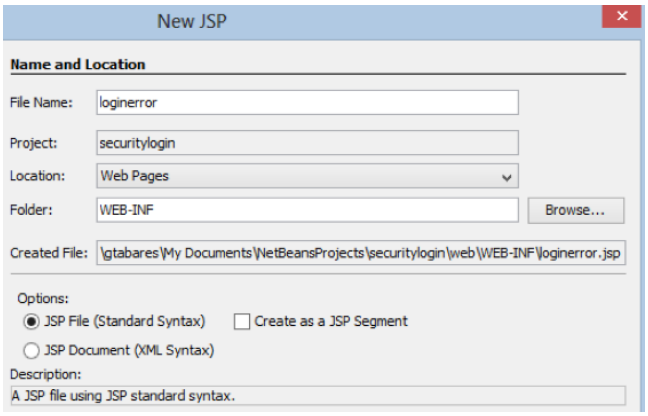


Imagen 8. Página de loginerror.jsp, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Y creamos en siguiente código del archivo loginerror.jsp.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Error de Login</title>
    </head>
    <body>
        Login Incorrecto <h1>!</h1>
        <br>
         <form method="POST" action="j_security_check">
            <table cellpadding="0" cellspacing="0" border="0">
                <tr>
                    <td aling="right">Nombre Usuario: </td>
                    <td><input type="text" name="j_username"></td>
                </tr>
                <tr><td align="right">Contraseña: </td>
                    <td><input type="password" name="j_password"></td>
                </tr>
                <tr><td></td>
                    <td><input type="submit" value="Login"></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Imagen 9. Código loginerror.jsp, código creado por el autor desde el IDE Netbeans
Fuente: Propia

A continuación configuraremos el Deployment Descriptor, Descriptor de Implementación XML de nuestro proyecto.

Para esto damos clic con el botón derecho sobre el proyecto, new Other …

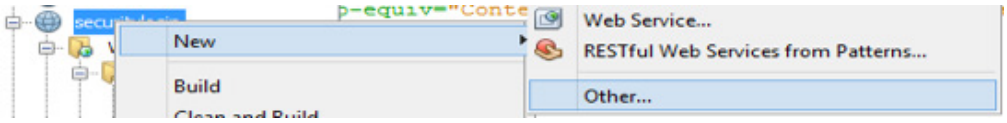


Imagen 10. Nuevo archivo Other, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Del cual seleccionamos la categoría web y el tipo de archivo Standard Deployment Descriptor (web.xml)

Encargado de describir como se despliega una aplicación web.

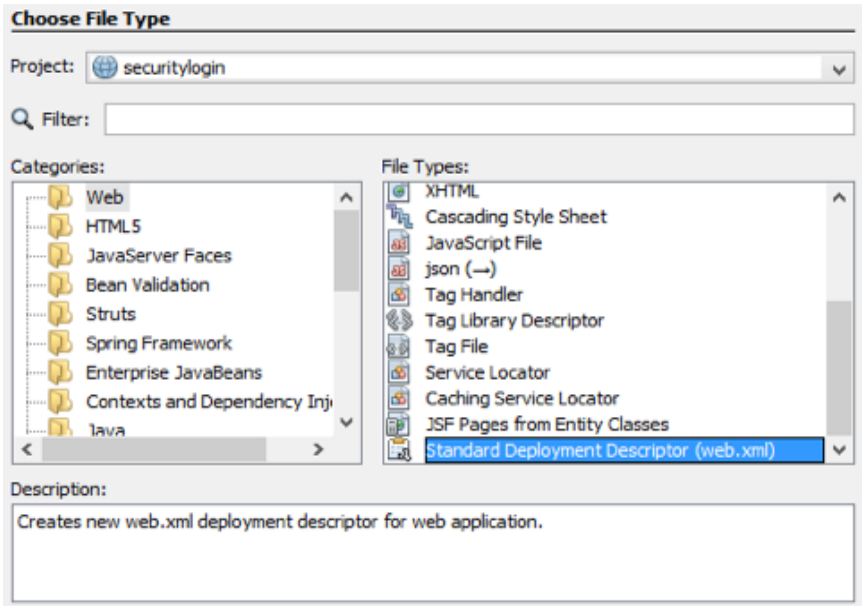


Imagen 11. Creación del archivo Standard Deployment Descriptor (web.xml), imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Nos crea el siguiente código, con una barra superior en la cual seleccionamos "Security".

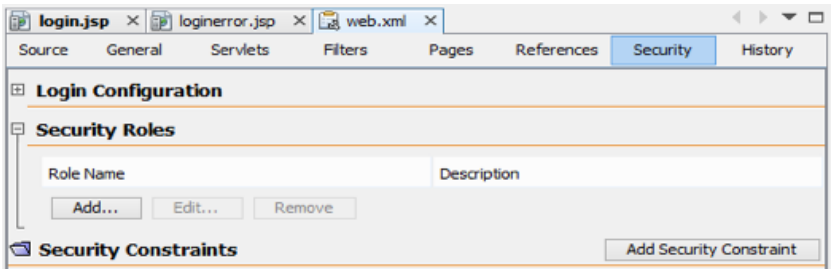


Imagen 12. Pestaña Security web.xml, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Desplegamos la sección de Login Configuration, y seleccionamos el tipo de autenticación que vamos a usar en nuestro caso es Form, indicamos la página del Form Login Page, el Form Error page y el dominio de seguridad que incluye GlassFish que es file.

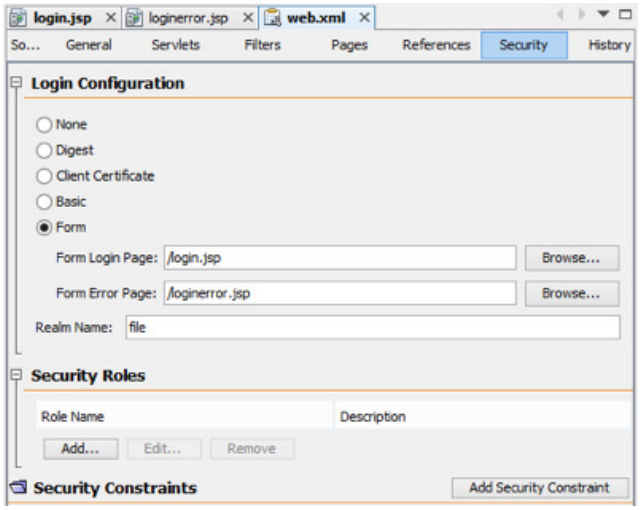


Imagen 13. Configuración del login, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Se pueden agregar nuevos roles de seguridad dando clic en el botón "add", crearemos un para administradores y otro para usuarios

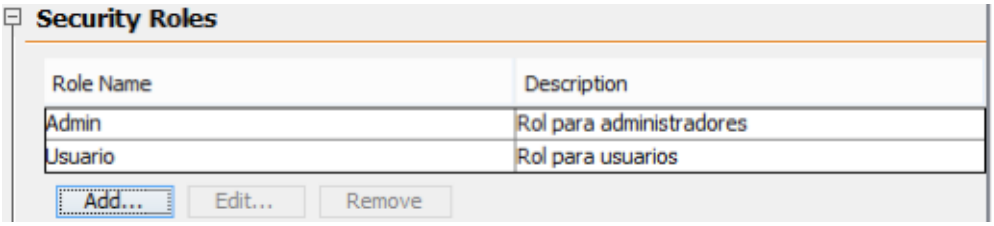


Imagen 14. Creando nuevos roles, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

A continuación debemos especificar que roles tienen acceso a qué páginas, para eso damos clic en el botón Add Security Constraint y clic en el botón Add…

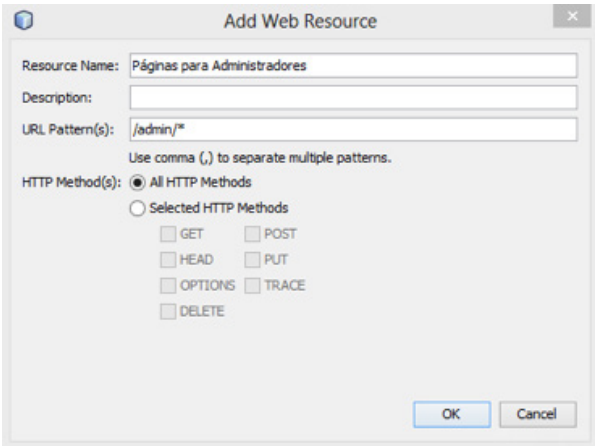


Imagen 15. Definición de páginas para administradores, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Para lograr que solo los usuarios autorizados puedan ingresar a las páginas autorizadas, es necesario activar la caja de checheo Enable Authentication Constraint y escribimos el nombre de los roles que van a estar autorizados para ingresar a la página, debemos pulsar en el botón edit, seleccionamos el o los usuarios y adicionamos a la derecha y clic en el botón OK.

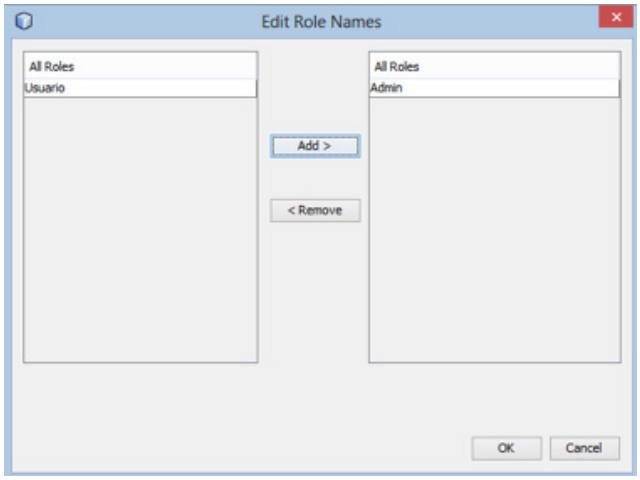


Imagen 16. Usuarios con acceso autorizado, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

El resultado final se muestra en la siguiente figura.

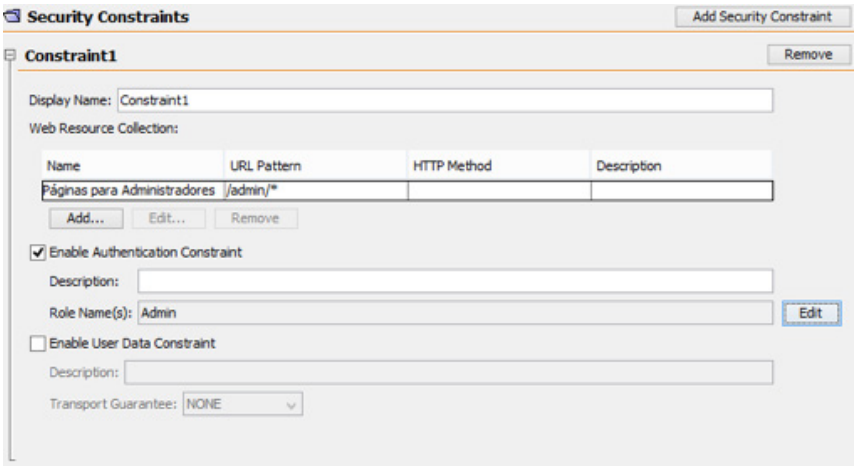


Imagen 17. Configuración del Constrain1, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Para el correcto funcionamiento es necesario crear una carpeta Admin con archivos, para que puedan ser accedidos por los usuarios administradores.

A continuación vamos a crear un Deployment Descriptor, seleccionamos el proyecto y con el botón derecho del mouse seleccionamos new file y en categoría seleccionamos GlassFish y en tipo de archivo GlassFish Descriptor.

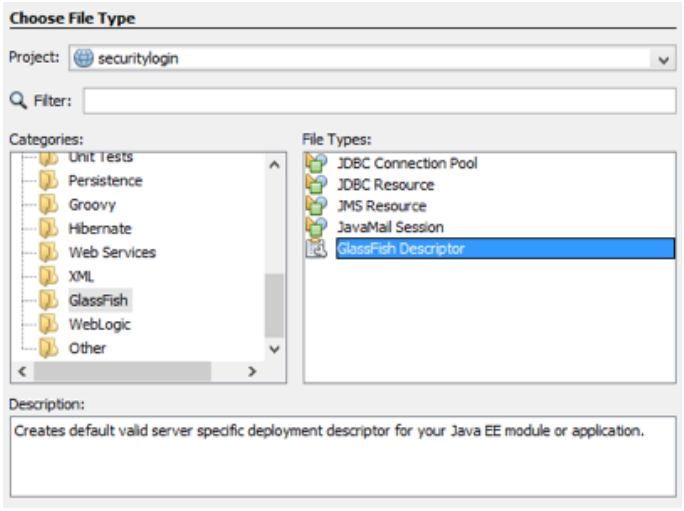


Imagen 18. Creación del archivo GlassFish Descriptor, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Dejamos valores por defecto y nos crea en la carpeta Configuration Files el archivo sun-web.xml

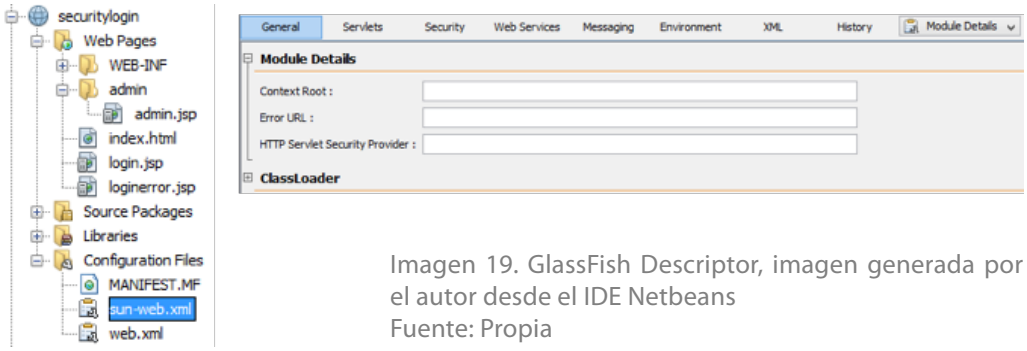


Imagen 19. GlassFish Descriptor, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Seleccionamos la pestaña Security e ingresamos en el campo Security Role Name: el primer rol que hemos creado, el cual se llama "Admin", para luego asignar grupos debe coincidir con el nombre que está utilizando el dominio de seguridad de nuestra aplicación en nuestro caso es "appadmin" y clic en el botón ok.

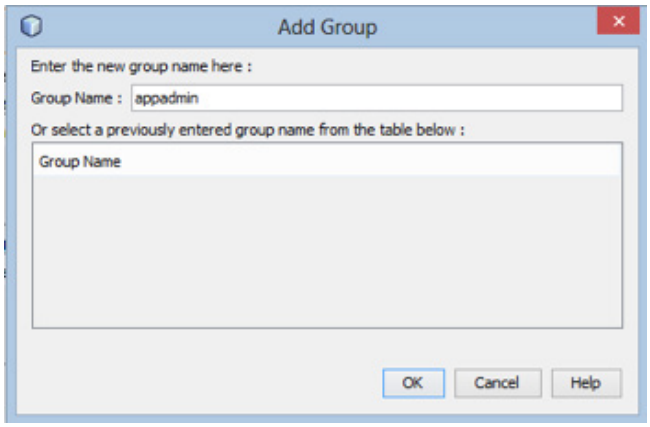


Imagen 20. Definiendo grupos, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia

Ya tenemos configurado nuestro sistema de seguridad de la aplicación, ahora es necesario crear algunos usuarios en el dominio para que puedan ingresar a la aplicación a través de la consola de GlassFish.

Consola de Glassfish

Para esto vamos a la pestaña Services, seleccionamos la opción servidores GlassFish Server con el botón derecho del mouse y seleccionamos View Domain Admin Console.

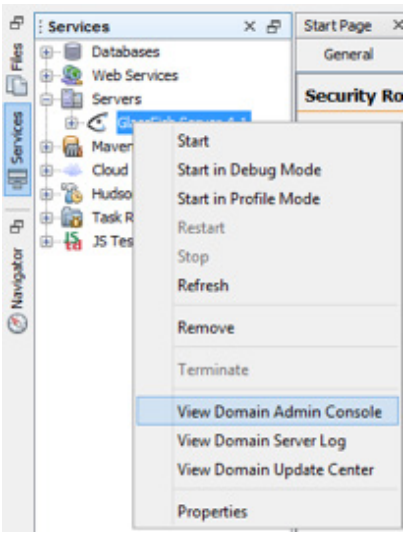


Imagen 21. Abriendo la consola de administración de GlassFish, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Se abre la consola de administración del servidor GlassFish en el navegador.

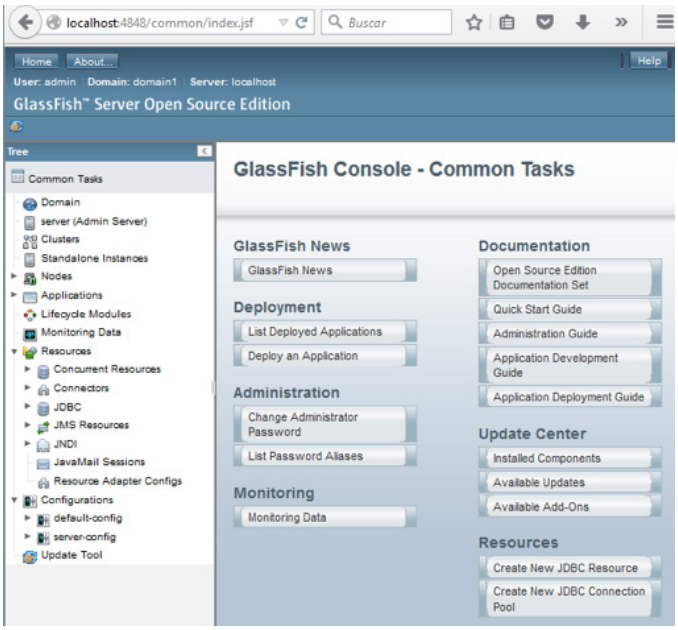


Figura 22. Consola de administración del servidor GlassFish, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Accedemos al menú izquierdo de la administración en la sección Configurations, default-config, Segurity, Realms, file.

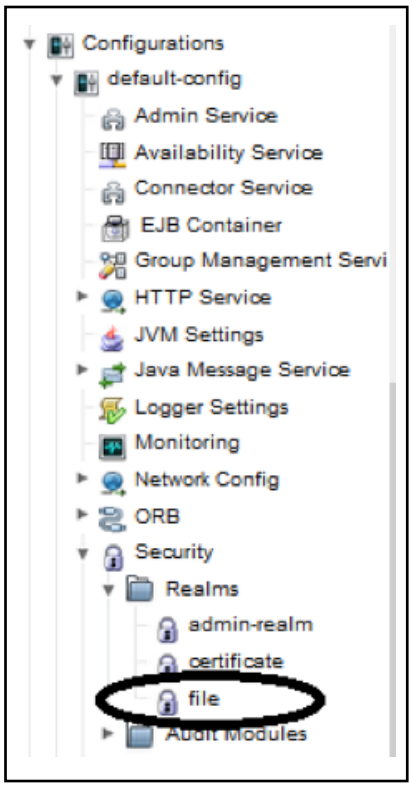


Imagen 23. Archivo de dominio de GlassFish, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Luego añadimos usuarios para acceder a nuestra aplicación, pulsando sobre el botón Manage Users.

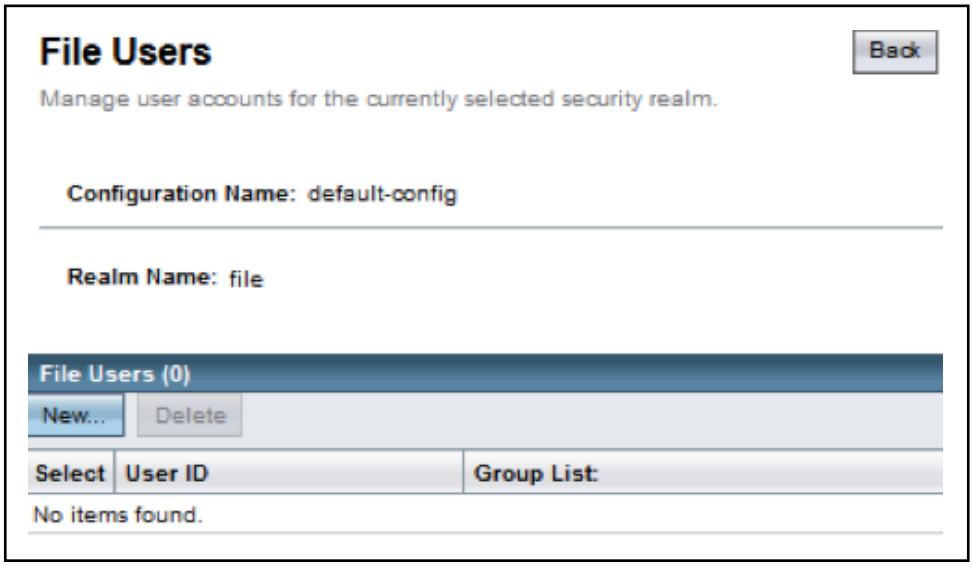


Imagen 24. Crear usuarios bajo el dominio file de GlassFish, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Pulsar sobre el botón New…. Para crear usuarios con su clave, bajo el dominio file. Crear el siguiente usuario y pulsar el botón ok.

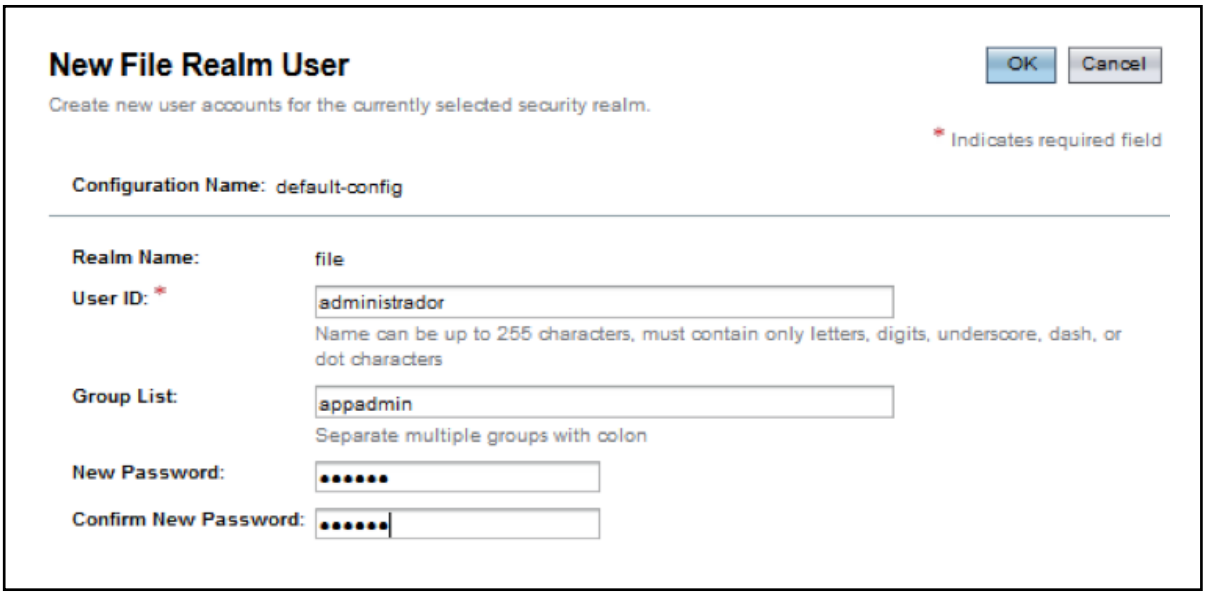


Imagen 25. Creando el primer usuario, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Ejecutar la aplicación y probar la validación del usuario administrador.

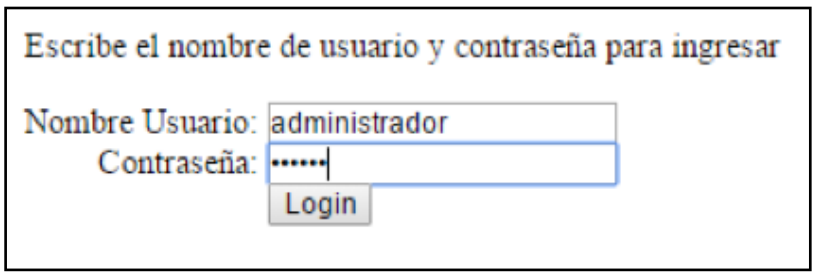


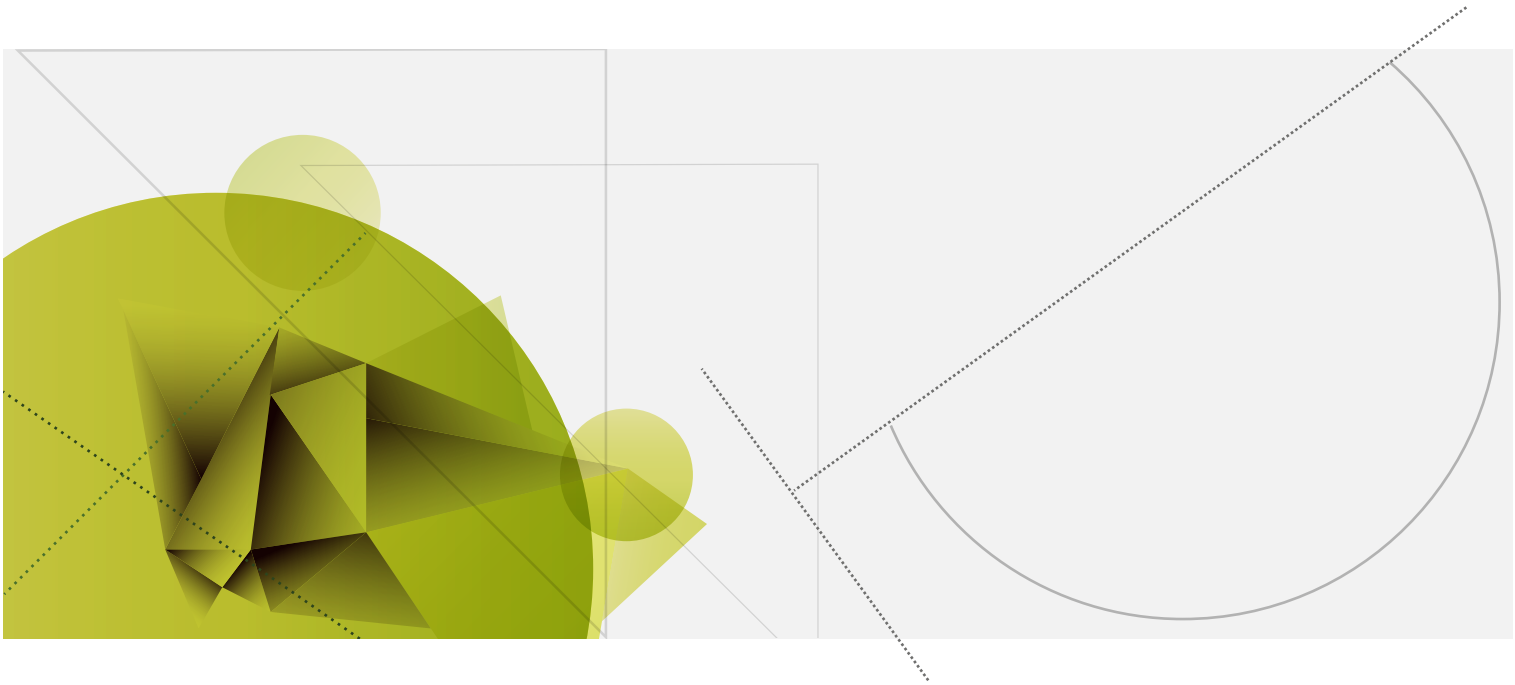
Imagen 26. loguin de la aplicación, imagen generada por el autor desde el IDE Netbeans
Fuente: Propia.

Pulsar el botón Login para acceder a la página de administración de la aplicación.

Bibliografia

- Azpitarte, R. et al. (2009). Introducción a la Programación Orientada a Objetos con Java. Departamento de Sistemas Informáticos y Computación Universidad Politécnica de Valencia.
- Balagurusamy, E. (2012). Object Oriented Programming with C++.
- Cake software Foundation. (2015). Entendiendo el Modelo - Vista - Controlador.
- Cardedo C. (2013). Tutorial de instalación y configuración de netbeans.
- Caules, C. (2013). Arquitectura java.
- EcuRed. (2015). Protocolo simple de acceso a objetos.
- Global Mentoring. (s.f). Instalación de JDK, Eclipse, ClasssFish y MySql.
- Infosec Institute. (2012). OWASP top 10 tools and tactics - info Resources.
- Lamarca, M. (2013). Aspectos tecnológicos de Internet.
- Martínez, E. (2014). Tutorial de JSP (4) – Scriptlet.
- Microsoft. (2007). Procedimientos de seguridad básicos para aplicaciones Web.
- Microsoft. (2015). Visual Basic.
- Microsystems. (2015). GlassFish.
- Purdum, J. (2012). Beginning. Object-Oriented Programming with C#.
- Rodríguez, A. (2015). Aprender programación java desde 0.
- Ruiz, M. (2013). Introducción a los Sistemas de Base de Datos.
- Soft, A. (2015). Programación: Cómo acceder a MySQL con JSP, Tomcat y JDBC.
- Swlibre12. (2015). Servidores Web Apache (linux y windows).
- Tejedor, R. (2008). Desarrollo de sitios Web dinámicos.
- Valdés, D. (2007). Los diferentes lenguajes de programación para la web.

Esta obra se terminó de editar en el mes de noviembre
Tipografá Myriad Pro 12 puntos
Bogotá D.C,-Colombia.



AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO